UNIVERSITY OF CALIFORNIA
SANTA CRUZ

BIRDSEYE: AN AERIAL ROBOTICS PIPELINE FOR RAPIDLY LABELED IMAGE DATASETS

A thesis submitted in partial satisfaction of the
requirements for the degree of

MASTER OF SCIENCE

in

ELECTRICAL AND COMPUTER ENGINEERING

by

T. Luca Altaffer

June 2024

The thesis of T. Luca Altaffer
is approved:

Table of Contents

List of Figures

List of Tables

Abstract

BirdsEye: An Aerial Robotics Pipeline for Rapidly Labeled Image Datasets

by

T. Luca Altaffer

In order to harness the potential of deep learning and computer object recognition in practical environments, a substantial collection of interesting features is essential. However, the creation of labeled image datasets is a significant challenge, hindering the broader adoption and autonomy by farmers, land managers, and ecologists.

To address this challenge, we present a data collection, annotation, and processing pipeline utilizing Unmanned Aerial Vehicle (UAV) based optical sensing. BirdsEye empowers non-deep learning experts to train, maintain, and deploy sophisticated computer vision methods on their own local land environments by substituting the need for aerial image feature identification with terrain based observation by subject matter experts. These annotated ground observations are then used to identify relevant image sections within UAV captured imagery.

By facilitating the rapid generation of labeled datasets, our approach can identify and characterize a diverse array of land and plant conditions. This has significant applications in areas such as disease monitoring, vegetation and pest identification, and precision treatments.

Acknowledgments

Thank you to my committee members: Steve McGuire, Colleen Josephson, and Jason Eshraghian.

I would also like to give a special thanks to my current and former colleagues in HARE Lab...

to Steve McGuire for giving me a life changing opportunity,

to Morgan Masters for being a kind mentor, a wonderful collaborator, and for permission to use many beautiful graphics,

to Adam Korycki for sharing academic adversities with me,

to Nicholas Kuipers for being a terrific friend and ally,

to Mathew Rochford for giving me confidence, humor, and reassurance,

and to Nikolass Bender for being my first technical leader and for giving me encouraging words in difficult times.

Chapter 1

Introduction

1.1 Problem Statement

With the global population increasing year by year, there is a need to increase the efficiency of local food supplies. According to Sylvester [43], by 2050, when the global population exceeds 9 billion, field agricultural production must increase by 70%. Furthermore, due to waste and exploitation of fertilizers, irrigation water, and other chemicals, increasing agricultural activity can compromise environmental sustainability and profits [18]. With agriculture consuming most of the world's land, there is a dire need to enhance intensive food production while optimising our land and water usage. While many practices exist that micromanage resources, such as consistent water use, on a larger scale, they begin to lose viability for the purpose of achieving high output while limiting environmental damage [18]. Precision Agriculture (PA) leverages technology to sustainably provide for a changing world.

Precision agriculture is an observational technique that measures and responds to temporal and spatial variability to improve agricultural production and sustainability [28]. Another, more popular, definition given by Pierce and Nowak [37] states, “Precision Agriculture is the application of farming strategies and methodologies to do the right thing, in the right place and at the right time,” where technology and the data they produce from detections decide what is right. PA is ultimately a data driven strategy for obtaining, analyzing, processing, and managing farm data to improve resource management and manage/plan crops and soil. In practical field robotics, PA is an ever-growing method that necessitates multi-modal sensor information to extract features such as crop volume, crop health, and pest and pathogen identification. There is an amalgamation of robotic systems that are used to approach this sort of work. Land-based robots are frequently used for specialized operations such as seeding, planting, weeding, treating, pruning, picking, handling, and harvesting, while on the other hand, aerial-based robots excel in monitoring and data collection [18]. Autonomy is a critical capability in both modes of robotics-based PA. These autonomous systems tend to fall into four categories [18]:

1. Navigation: The vehicle moves without an operator, traversing independently in a predetermined course.

2. Sensing: The vehicle measures, detects, or samples any relevant information.

3. Action: Necessary actions, such as treatment, are enacted by the vehicle in a dedicated zone.

4. Mapping: The vehicle, generates maps for further use in Geographic Information Systems (GIS) from field features.

Aerial robots, such as drones, have been becoming ever more prevalent in this field. Unmanned Aerial Vehicles (UAVs) have the advantage over ground vehicles of being able to navigate difficult-to-traverse locations and remotely sensing over wide areas for crop and soil monitoring. Typical uses of remote sensing are based on the collection of aerial-based images at various spectral ranges. Within this collection of aerial images, desired features can be better understood and extracted for analysis. Datasets can be built from this data, and deep machine learning techniques can be applied for the autonomous detection of physical symptoms that indicate diseases or other forms of stress. This pipeline to enable autonomous crop monitoring from data processing is a cornerstone process of PA.

Supervised learning, a deep learning technique commonly employed in PA, relies on manually labeling data, a tedious, time-consuming, and expensive endeavor. It is an unfortunate bottleneck that prevents the general scientific community access to deep computer vision and the benefits it could yield to PA research. This work presents BirdsEye, a pipeline to rapidly generate custom, semantically-annotated geographic image datasets for vision-based tasks. Annotations are labeled in the field at ground level, by agro-ecological experts, where then physical positions are related to pixels in aerial image streams. This correlation is accomplished via calibrated cameras and precise positional information. BirdsEye helps bridge the gap between farmer and autonomous system, bringing their unique insights and expert knowledge into their own custom trained computer vision system.

1.2 Approach

This work presents a data collection and annotation to processing pipeline for arbitrary vision-based tasks where aerial images are used to identify ground phenomena. BirdsEye takes in drone-based imagery and its positional information, as well as in-field GNSS based labels, to generate annotated datasets. Not only does this approach cover the development of BirdsEye as a data processing pipeline, but it also covers the robotic systems necessary for data collection and monitoring.

The drone collects imagery via a custom payload package that records multi-spectral and spatial data. Several sensors are integrated to do this, including three camera systems with eight total lenses, an inertial measurement unit (IMU), a radio altimeter, and a GPS receiver. These sensors communicate and save data through the onboard computer via an event-service framework known as Robot Operating System. After a manual flight or autonomously executed waypoint mission, the data are extracted from the drone and processed locally through BirdsEye. After organizing and processing the data, BirdsEye then conducts spatial rotations to relate the imaging plane to the real world. With this, field based labels can be projected into the image frame, allowing for seamless image annotation.

For this pipeline to be successful, error from the mathematical camera modeling and data from the onboard sensors must be minimized. Much of the effort of this

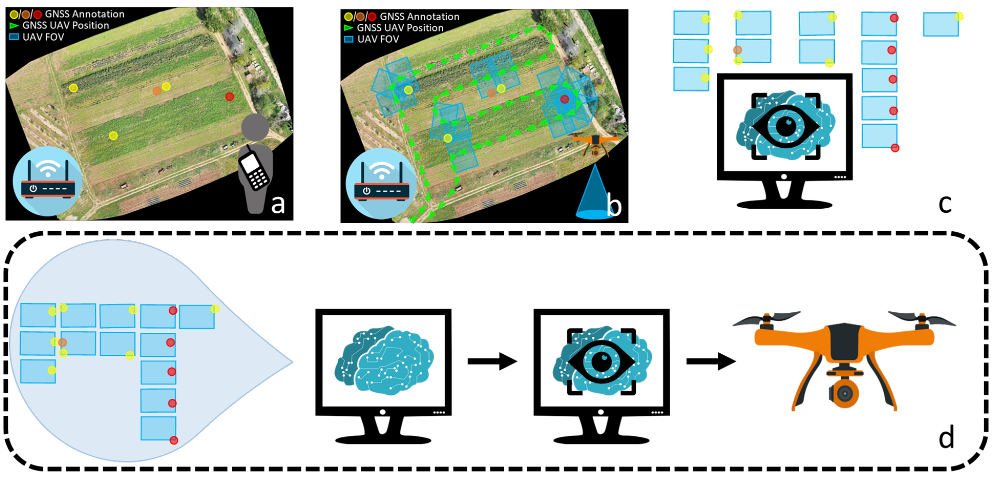


Figure 1.1: **a:** Field workers identify and label features-of-interest with handheld RTK enabled GNSS devices. **b:** UAV-based images are taken and associated with designations in view via BirdsEye. **c:** A software backend stores the associations in a database to support deep CNN training[2].

work's approach deals with this minimization of the system wide error. Our goal is to have an average error of no more than 6 cm in our spatial reprojections, meaning that BirdsEye should reproject our in-field labels very close to the ground-truthed locations. Our 6 cm goal comes from theoretical sensor error margins and necessary range to have accurately placed lables. To validate system performance, we conducted a flight over Apriltags, which serve as our ground locations, and placed in-field labels over the Apriltags. Apriltag centers are spatially compared to our in-field label locations, and the difference between them is easily calculated both in image space with error in pixels and in real-world space with error in meters.

[2]Figure developed by Morgan Masters.

1.3 Scope of Thesis

The goal of this thesis is to demonstrate a proof of concept for the BirdsEye pipeline. From the data collection stage to data processing, we are expecting to see spatial image-to-world projections and world-to-image reprojections with minimal error. Much of this work entails identifying and mitigating hardware restrictions that prevent us from obtaining desired results and error margins.

This thesis is organized as follows: Previous Work, Methodology, Experiments/Results, Conclusion, and Future Work. The Previous Work section will go in-depth on UAVs in PA, the spectrographic response of plants, and deep computer vision techniques. Methodology will discuss the systems, how they work, their challenges, and the general approach. Experiments and Results goes over the validation experiments and the data generated. The Conclusion will outline BirdsEye and the system's viability to be used for deep learning applications. Finally, Future Work will present the next steps of this work and how it can be applied further.

Chapter 2

Previous Work

This section outlines several general concepts and the related work associated with robotic systems design and the current state of the art in precision agriculture. In later sections the system design and the experimental knowledge will be discussed, with corresponding complications and mitigations highlighted.

2.1 UAVs in Precision Agriculture

In farms, different agricultural activities have been carried out by UAVs. Among them, UAVs are being increasingly used for disease identification and stress monitoring [33]. Aboard these UAVs is an amalgamation of cameras, sensors, motors, rotors, antennas, receivers, and controllers. These systems are becoming critical tools in agriculture, allowing farmers and growers to monitor their crops over wide areas more effectively while keeping costs low [21]. Primarily there are two types of UAV platforms that are used in agriculture: rotary-wing (helicopter, quad-copters, hexa-copters, octocopters, etc...), and fixed-wing. Fixed-wing UAVs are typically larger than rotary-wing drones and are used for rapid survey coverage over large areas [25]. For the purpose of plant monitoring, both styles of UAVs have been used. The choice between each platform usually depends on the area needed to be covered and the payload being used. Fixed wings, for example, are typically faster, can cover a larger area, but can't carry a heavier payload without additional infrastructure for takeoff and landing. Rotary-wing drones, excel in confined or congested spaces with a heavier payload and slower speed. They can hover when necessary and are easier to control. Fixed-wing UAVs are considered to be more cost effective in the field due to the total field coverage per flight duration they are able to achieve. For the purposes of this work, we are using a quad-copter rotor-wing drone due to our heavy lift necessity and desire to minimize launch and landing space requirements.

One of the basic uses of UAVs is to capture images across the electromagnetic spectrum. Agricultural-specific cameras typically can capture from frequency response regions ranging from the ultraviolet to the infrared. Comparing between spectral responses in specific regions (potentially beyond the human visible range) can help us assess the physical condition of plants in real-time under field conditions [23]. With this information, we can monitor and visualize physical plant features that are otherwise invisible to the naked eye. The information contained in these images can be later extracted and transformed into useful information in an image processing step and with deep learning tools [47].

Figure 2.1: Image of our quad-copter drone (a DJI Matrice M300) with heavy lift capabilities hovering in the field.

2.2 Plants & Irradiance

As plants receive the sun's radiation, natural compound pigmentations such as chlorophyll and xanthophyll absorb a majority of the radiance of the visible band but reflect most radiance at the near infrared (NIR) (780nm -1200nm) and red-edge (670nm - 760nm) wavelengths [48]. While spectral responses differ for various species of plants when they become unhealthy, their spectral responses change as the biomass begins to exhibit physical symptoms. Plant stress and disease are generally identified by observing the physiological disturbances caused by foliar reflectance in a near-infrared (NIR) portion of the spectrum. NIR imaging is conveniently accomplished through the use of commodity CMOS imagers (which are sensitive to a broad range of electromagnetic spectrum extending beyond the visual) with an appropriate filter to exclude the visible range. There are many causes that can contribute to a plant undergoing unhealthy stress such as disease, temperature, water stress, or pests. Utilizing multi-spectral imaging (that is, imaging at several specific frequency bands), one may monitor these stress induced physical responses. Cameras such as the Micasense Rededge MX are industry staples for sensing plant health as they capture both the red-edge and NIR wavelengths.[1].

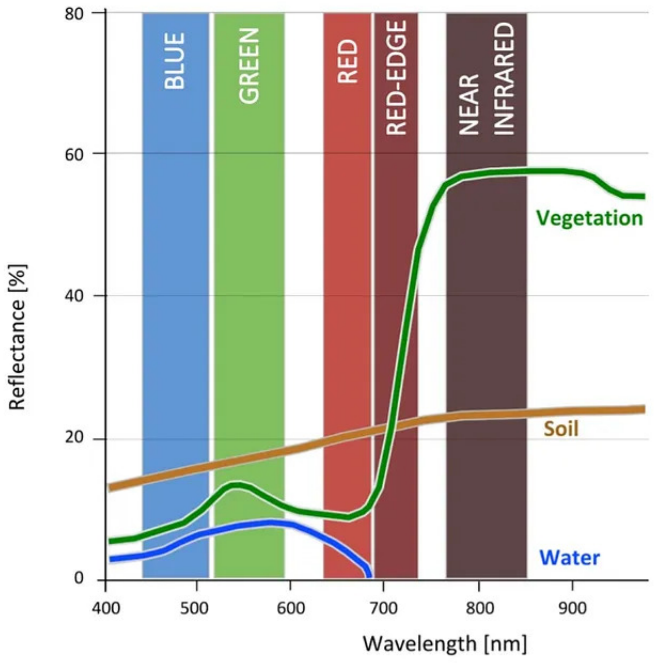


Figure 2.2: Spectral reflectance of vegetation vs soil and water. Vegetation reflects most of the red-edge and NIR wavelengths making it easier to see them at these wavelengths. [42]

In practice, this kind of imaging technique is used in a wide range of crop monitoring: stress in orchards [48], water stress-induced lettuce [35], and water deficit stress in rice crop [28]. However, these experiments are terrestrial-based which, outside of controlled environments, come with several complications. Ground based imaging is time consuming and can be difficult to obtain wider perspective shots on larger crops. Land robots, or unmanned ground vehicles (UGVs) such as a four-wheeled Husky[1] or a Unitree B1 quadruped [2] can have a difficult time navigating through furrows without damaging crops. Based on these limitations, UAVs, remain the preferred choice to image plants over wide areas.

2.3 Deep Learning

2.3.1 Digital Image Representation

Before introducing deep learning techniques, it is important to first illustrate how images are described in the digital world. Images are a 2D array of pixels, where pixels are the smallest possible unit that makes up an image. These pixels are assigned values which make up the image as seen below with Figure 2.4. For this example, the pixel values are normalized between 0 and 1; in a standard 8-bit image the values would be between 0 and 255. A pixel value of 0 corresponds to blackness while 255, or 1 in this case, whiteness. As this is a gray scale image, anything between 0 and 1 appears as a gray pixel with shifting intensity.

For color images, three arrays are 3D matrices instead of 2D. Color images share the same height and width elements of a grey scale image but have three channels

[1]`https://clearpathrobotics.com/husky-unmanned-ground-vehicle-robot/`
[2]`https://shop.unitree.com/products/unitree-b1`

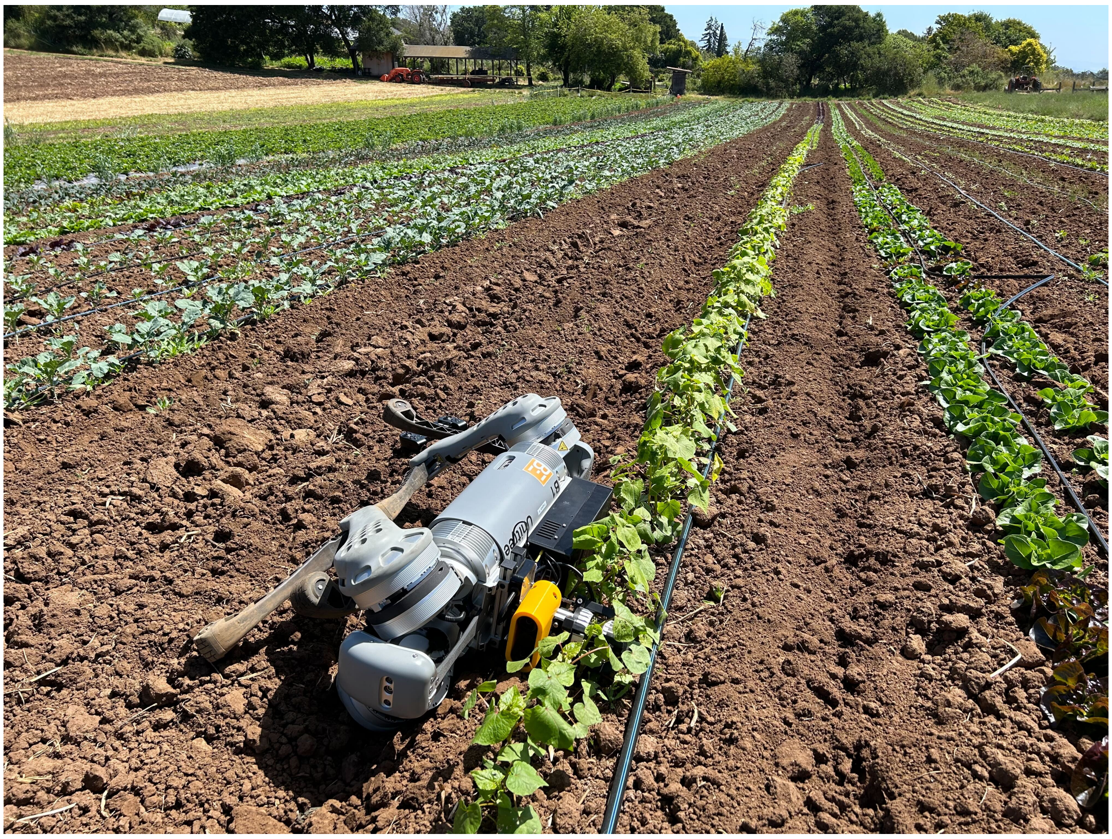
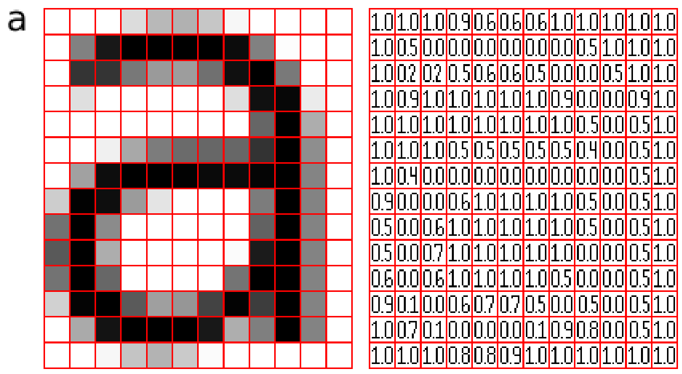


Figure 2.3: Photo of a Unitree B1 quadruped robot fallen over after attempting to navigate through soft dirt. Terrestrial based robots can struggle to navigate through farms due to restrictions such as dirt texture, furrow spacing, and a high risk of damaging crops.

for red, blue, and green (RGB). As experimented by physicist Thomas Young, the human sensation of color can be produced using these three wavelengths. For example, we can represent the color in a matrix as:

$$(\lambda_r, \lambda_g, \lambda_b) = (650, 530, 410)\text{nm}$$

The three pixel values at each location combine to form the color you see an the image. Figure 2.5 gives a visualization of the 3D array representation of color images.



1.0	1.0	1.0	0.9	0.6	0.6	0.6	1.0	1.0	1.0	1.0	1.0
1.0	0.5	0.0	0.0	0.0	0.0	0.0	0.0	0.5	1.0	1.0	1.0
1.0	0.2	0.2	0.5	0.6	0.6	0.5	0.0	0.0	0.5	1.0	1.0
1.0	0.9	1.0	1.0	1.0	1.0	1.0	0.9	0.0	0.0	0.9	1.0
1.0	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.5	0.0	0.5	1.0
1.0	1.0	1.0	0.5	0.5	0.5	0.5	0.5	0.4	0.0	0.5	1.0
1.0	0.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.5	1.0
0.9	0.0	0.0	0.6	1.0	1.0	1.0	1.0	0.5	0.0	0.5	1.0
0.5	0.0	0.6	1.0	1.0	1.0	1.0	1.0	0.5	0.0	0.5	1.0
0.5	0.0	0.7	1.0	1.0	1.0	1.0	1.0	0.0	0.0	0.5	1.0
0.6	0.0	0.6	1.0	1.0	1.0	1.0	0.5	0.0	0.0	0.5	1.0
0.9	0.1	0.0	0.6	0.7	0.7	0.5	0.0	0.5	0.0	0.5	1.0
1.0	0.7	0.1	0.0	0.0	0.0	0.1	0.9	0.8	0.0	0.5	1.0
1.0	1.0	1.0	0.8	0.8	0.9	1.0	1.0	1.0	1.0	1.0	1.0

Figure 2.4: Digital image representation magnified 16 times using pixel doubling [2]

Every subsequent concept in computer vision relies this concept of of treating images as arrays of pixel values.

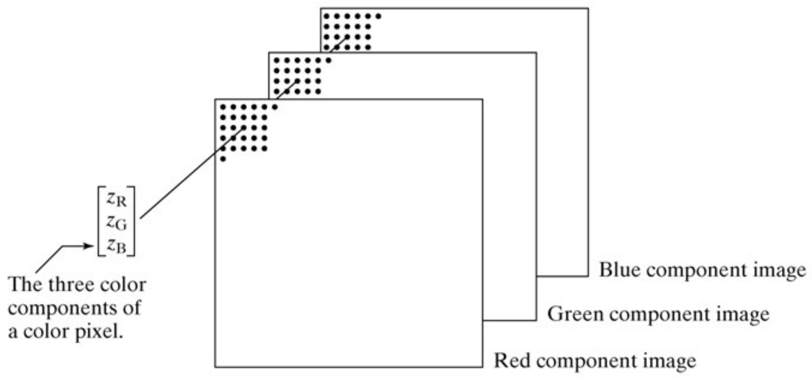


Figure 2.5: Representation of Digital Color Image as a 3D array. This representation allows matrix math to be applied over many channels. [34].

2.3.2 Convolutional Neural Networks

A Convolutional Neural Network (CNN) is a deep learning algorithm that uses input images, takes learnable weights and biases to assign importance, then differentiates images. They are a variation of Neural Networks (NNs) where instead of neurons being 1D in nature, CNNs have 2D layers. For this reason, CNNs remain the preferred choice when solving image related problems. The subsequent subsections present a brief overview of a CNN's functions.

2.3.2.1 Neural Networks

Although there is a many difference between NNs and CNNs, due to their layer composition, there are also many similarities in how they are constructed and trained. CNNs are a subclass of NNs, thus discussing NNs prior to CNNs is the logical progression.

NNs function by having layers of nodes, or artificial neurons, an input layer, one or more hidden layers, and an output layers [14]. A given node sends data to the next layer of the network when it is excited by having its output exceed a given threshold value.

Each individual circle in Figure 2.6 is a node and the arrows are weights. Each nodes functions as a linear regression model comprised of input data (x_d), weights (θ_d), a bias (I), or threshold, and an output (z):

$$z = I\theta_0 + x_1\theta_1 + x_2\theta_2 + ... + x_d\theta_d$$

Weights are assigned after the input layer is sampled, helping to determine the importance of every variable. After the inputs are multiplied by their weight and summed together, they are passed through an activation function, which determines the

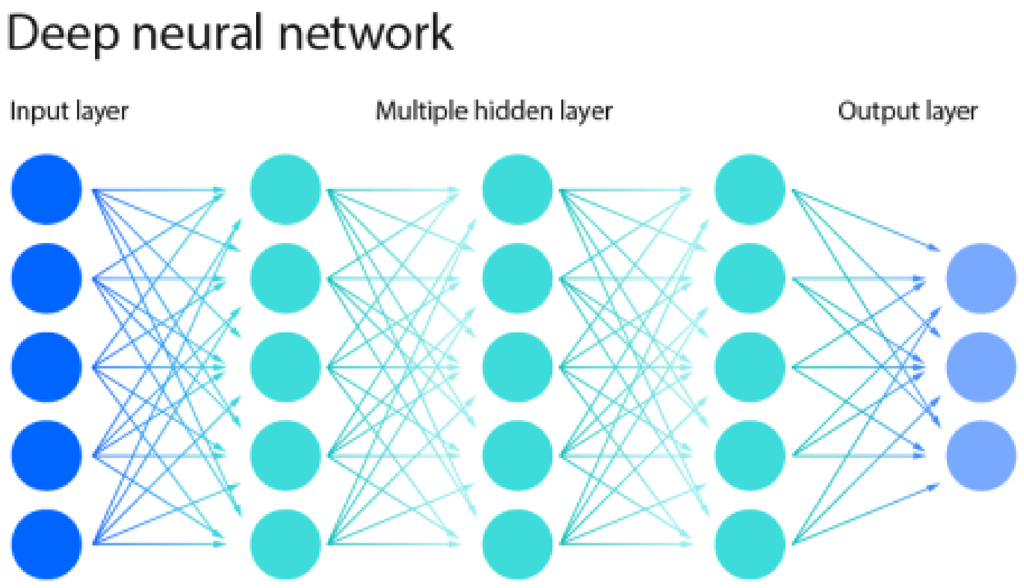


Figure 2.6: Architecture of a simple Neural Network. The term "deep" refers to the "depth" or quantity of the hidden layers. Typically three or more hidden layers is considered deep. Input layers are sent through the many hidden layers which then produce a final output [14].

output. The value of this output determines if it excites the node, which passes data to the next layer as its input. The output layer also serves to classify the data, where each output node corresponds to a particular class, with hidden layers allowing for more complex classification tasks.

2.3.2.2 CNN Model Architecture

Similarly to NNs, the architecture of CNNs are analogous to neuron connectivity in the human brain. They are designed to understand more complexity and details in images compared to NNs. This is due to their ability to capture spatial and temporal dependencies by applying specific filters. They also reduce the amount of parameters by re-using weights to preform a better fitting to the image datasets [40]. A key part of this technique is to make images easier to process by reducing the image resolution while attempting to not lose essential features. This is critical in training a network to have great learning capabilities while also being scalable to larger sets of data.

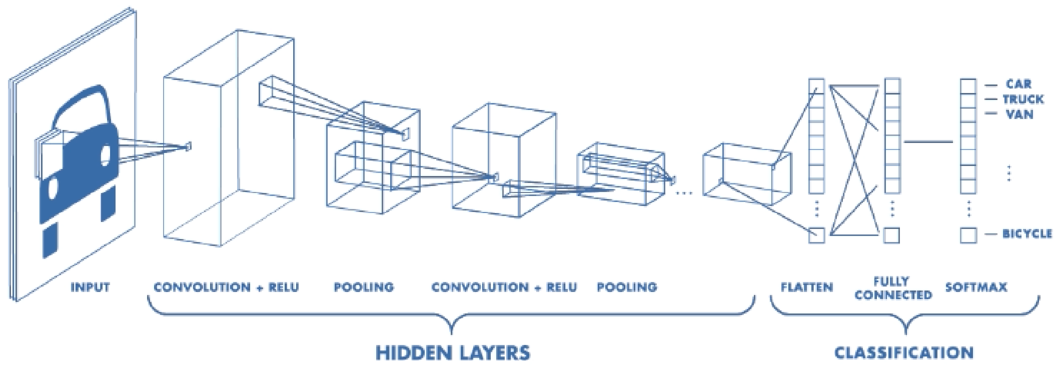


Figure 2.7: Architecture of a basic Convolutional Neural Network. An input image is passed through the hidden layers comprised of a convolution, activation function, and pooling step. It is repeated several times before going into the classification step where the hidden layer output is flattened then plugged into a fully connected network which is then given to a softmax layer. [40].

After input images are given, the CNN's first layer is a convolution layer. This convolution layer is at the core of CNNs and they often have many of these hidden layers (convolution + pooling layer) making them "deep". Briefly, convolution applies a function (expressed as a matrix) to another matrix. For CNNs, a 2D filter is convolved around either an image or the output of the previous layer. This filter, or convolutional kernel, is a nxn 2D array with values determined during training. As seen in Figure 2.8, the filter values are multiplied to the image values, for a given neighborhood, via dot product then added to form a the output value (the purple square) in our resulting matrix. This process is repeated for all values in the area of the image as it moves by a stride, the amount of pixels the filter frame moves by. The output of this calculation is then inputted into an activation function, such as ReLU[15], which gives us the result of this convolution layer. Activation functions introduce non-linearity into the model, which permits the network to learn complex patterns and relationships in the data. Without them, networks would not be able to handle large volumes of complex data.

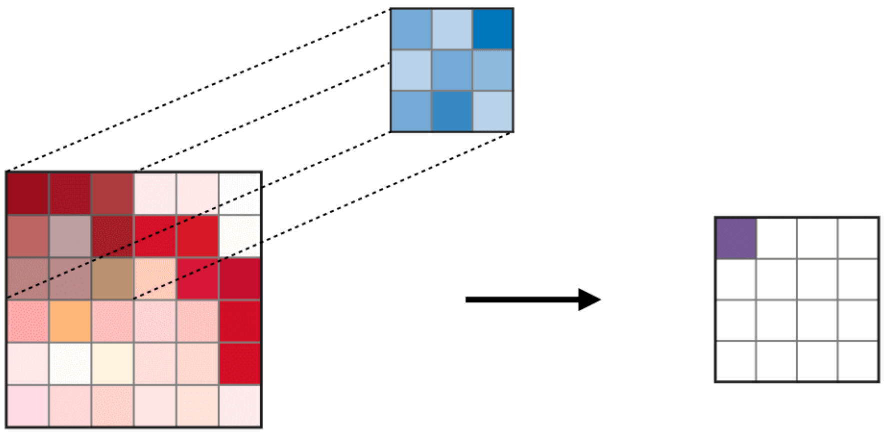


Figure 2.8: Visualization of a convolution step. Here the filter (in blue) is a $3x3$ matrix and the image (in red) is a $5x5$ matrix. After the convolution is complete the new image (in purple) size is a $4x4$ array [7].

Convolution steps are typically followed by either another convolution or a pooling layer. Max pooling is the most commonly used, however, other types of pooling exist such as average pooling. Pooling layers extract dominant features and reduce the array dimensions which saves computer power. While this can be done with any sized kernel, using the example in Figure 2.9, it divides the input area into a $2x2$ area and extracts these dominant features. For max pooling, it extracts the largest value within each $2x2$ cell.

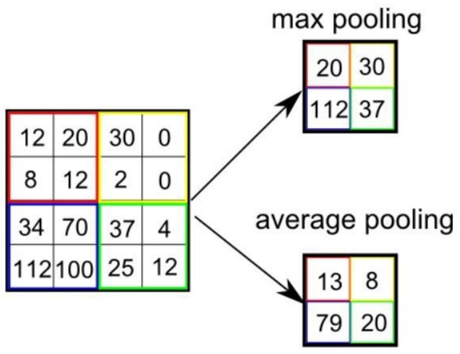


Figure 2.9: Visualization of a the pooling step. In max pooling, the largest value from each 2x2 submatrix is taken and placed into the new 2x2 matrix. In average pooling, the values of each 2x2 submatrix are averaged and used in the new 2x2 matrix [40].

The combination between input, convolution, ReLU activation, and pooling are essential ingredients for the feature learning selection of CNNs or unsupervised feature extraction. The second section of the CNN performs classification which is actually just a standard NN described previously. However, NNs only accept 1D inputs and the output from the feature learning section of the CNN is 2D. Thus, we must flatten the output with a flattening layer before handing it off as an input for classification as a 1D data array. Classification is performed using a softmax layer which outputs the neuron with the highest value as the class. The benefit of softmax rather than traditional max is because softmax rescales the output values to be between 0 and 1, where all the output values are summed together equal to 1. This allows values to also be interpreted as a prediction confidence rating. We can represent the softmax calculation as:

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

where z is a vector of raw outputs, z_i corresponding to a single neuron output vector while the summation of z_j corresponds to the sum of all the output vectors.

2.3.2.3 Model Training & Testing

A NN or CNN must be trained in order to recognize patterns, make decisions, and predict classifications from data. Data used for model training are called datasets and it is important that samples used in training datasets are not used to test the model. Having an independent dataset for testing is important to not invalidate results. Models should have the capability to classify unseen data and it is important to grade them to this standard.

Annotations in input images are ground-truthed, user-defined, classifications of particular pixels. In supervised learning methods, all training data contains labels which tells the model which class the data actually belongs to. Other types of learning methods such as semi-supervised and unsupervised employ a variety of techniques where some or no labels are used to train the model. There are several benefits to all of these strategies. In supervised learning, the algorithm learns to associate input to outputs by identifying patterns and relationships within the dataset. Unsupervised learning entails training with unlabeled data where the algorithm attempts to find hidden patterns or intrinsic structures within the dataset, exploring the data without oversight or guidance. In between both techniques, semi-supervised learning uses limited amounts of labeled data and a more extensive set of labels to improve the model's accuracy and performance [13].

Another important consideration is giving too much or too little training data. Models can be over- or under-fitted to their training data and lose their ability to generalize to unseen examples. For the over-fitting example, models can exhibit high training accuracy but when applied to a test dataset, they perform significantly worse. Here, the model learned to extract differences between the training data rather than general patterns used in recognition. Over-fitting can occur for reasons such as high variance and low bias, the model being overly complex, or the size of the dataset. Improving the quality of the training dataset or adding more unseen data to the dataset are viable strategies to reduce over-fitting. There are other strategies such as a dropout layer or early stopping which helps remove complexity from the network.

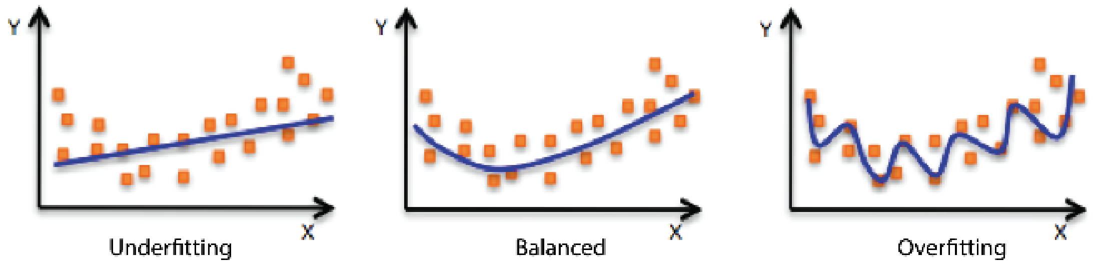


Figure 2.10: Visualization of network model fitting. Underfitting is too simplistic and struggles to capure the complexities of the model. Overfitting does too much to capture the model complexities and misrepresents the general trends [11].

In addition to training and testing datasets, validation sets are used to check performance after each training cycle. This allows for impromptu parameter tuning for aspects like filter size, filter count, and stride; however, the validation set isn't used for model learning. This makes validation sets and testing sets somewhat interchangeable as neither are represented in the training set. Often times, the validation and testing set were broken off from one dataset, with more data slightly weighing in favor for the testing set.

When models predict the sample classes correctly, there is still an error vector associated with the training sample. This vector is calculated based on the desired loss function (describing the difference between observed predictions and desired predictions) being used, which there are a lot to chose from. Cross-entropy is such a standard loss function. It is also called log loss, and can be expressed as the following:

$$loss(y_d, y_a) = -log|1 - y_d - y_a|$$

where y_d is the desired output and y_a is the actual output [4]. This is calculated for every entry in the output vector and increases as the predicted probability diverges from the actual label. As an example, having an observation label as 1, and the prediction label is 0.01 would be a bad result and lead to a high loss. In an ideal world, a model would have a loss of 0, however, this is impractical in reality. As the prediction probability approaches a value of 1, the logarithmic loss slowly decreases and rapidly increase as the prediction decreases. This means that the logarithmic loss affects both types of errors but especially the predictions that are confident and wrong [4].

After input images are fed through the whole model, the expected output is compared to the actual output and the loss is calculated. This loss is utilized to update the model's weights in a process known as back-propagation. This is based upon the principle of gradient descent, the main training algorithm in every deep learning model. Gradient descent is an iterative optimization algorithm for finding the minimum of an objective function [8]. In deep learning, this objective function corresponds to the loss function of the network. Gradient descent works by repeated looping over the process of taking the gradient, or the first order derivative, of the objective function with respect to the network parameters, then updating these parameters.

There are three types of gradient descent algorithms that one could choose based on the amount of training used in a single iteration of the algorithm. Batch training passes all of the training data in a single iteration of the algorithm. The average of the gradients are then taken and the parameters are updated using the computed average. This method can be very inefficient and time consuming if the dataset is large. In contrast, stochastic gradient descent is a technique which can be used where a single sample is used for a single iteration, a process with its own drawbacks. With just one sample per iteration, the loss function is not guaranteed to decrease in each iteration and will likely never reach the global minimum. Hence, mini-batch is most commonly used compromise between these two approaches. It is a method that contains a fixed number of training examples that are less than the full dataset. In each iteration, the network trains a group of samples until all samples of the dataset are used. Passing all of the training data through the network is called an epoch; the number of epochs is a hyperparameter that defines the number of times that gradient descent will pass through the entire dataset [8].

For each epoch, the full set of training samples is used for weight correction.

In general, the more training epochs used the better resulting network performance; however, more epochs can be expensive in terms of computation time. It is recommended to use as many epochs as time and hardware permits, whether 10, 100, 1000, or more based on the quantity of the training data.

One way to improve these gradient descent algorithms is by applying an adaptive learning rate. This adds another hyperparameter that controls how much to change the model in response to the estimated error each time the weights are recalculated. A learning rate that is too small may result in a long training process, whereas a value too large can lead to sub-optimal set of weights or an unstable training process. Choosing an appropriate learning rate can be a challenge [19]. Momentum is an adaptive learning rate technique often used that helps accelerate the gradient descent by adding a fraction of the previous update to the current update. As its name suggests, momentum has a "velocity" concept to parameter updates, where in steeper areas one may want to take smaller steps (lower learning rates) to avoid overshooting and in flatter areas it takes larger steps (higher learning rates) [5].

As stated previously, the test dataset is used to grade the model's performance. When testing the model, each testing sample is pushed forward through the model, excluding back-propagation as the model is done learning. After this process is done, the model makes a prediction on the input's class based on the max neuron value of the output layer. This prediction is then compared to the known ground truth label, as this dataset also contains annotations. After testing each sample in the test dataset the number of misclassifications is used to calculate the accuracy using the standard formula:

$$Accuracy = \frac{TotalSamples - Misclassifications}{TotalSamples} \times 100$$

2.4 Annotation Pipelines

Annotation pipelines are often used to create and label large annotated datasets. It often involves tedious manual labor as each frame must have appropriate labels. There are many different annotation methods that exist to create labels such as bounding boxes, polygon segmentation, 3D cuboids, key-point and landmark, and lines and spline to name a few. The purpose of this section will be to go over a few of these labeling techniques their use in existing pipelines.

One of the more common types of annotation methods are bounding boxes. This object detection technique involves creating a rectangular box that define the image-space location of a target object. This determines the 2D x, y pixel space location of the desired object. Thus, these bounding can be represented by their upper left coordinate and lower right coordinate. Like other object detection methods, bounding boxes utilize Region Proposal Networks (RPNs) to generate potential object proposals. These are then analyzed by feature extraction networks to accurately classify and localize objects in frame [29]. While useful for identifying and localizing generalized objects, bounding boxes can struggle with more detailed forms of feature and pattern recognition as they do not make labels at the pixel level [38].

Segmentation is another fundamental process in computer vision. This method entails breaking an image down into classified regions. Segmentation enables the identification and differentiation of objects by diving visual content into segments [29]. This occurs at the pixel level, thus outlines of these segments are easier to build. There are several types of segmentation (instance segmentation, polygon segmentation, etc...) but for the purposes of this work we are dealing with semantic segmentation. The difference between semantic segmentation compared to there form of segmentation is that it assigns class labels to each pixel in an image set. This form of classification allows for a much more detailed understanding of an object's boundaries and meaningful regions [29]. They are also often used alongside CNNs.

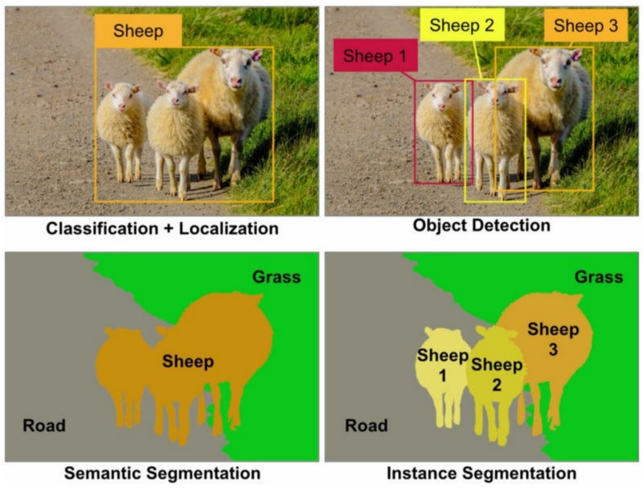


Figure 2.11: Segmentation techniques versus bounding boxes. Segmentation has the ability to get pixel level boundaries around specific classifications rather than bounding boxes which generalize objects inside a rectangle [31].

There are many annotation pipeline tools that are practically used. You Only Look Once (YOLO) is a widely used, open sourced, detection architecture that divides an image into a grid of cells and predicting bounding boxes for objects that are located in each cell [29]. YOLO is a very well trained neural network that has gained much popularity due to its user friendly simplicity and success in making accurate bounding boxes. Alternative object detection methods utilize separate classifications and localization stages while YOLO does both in a single neural network. This process speeds up computation and the detection process while also accurately localizing and classifying objects in real-time[29]. These features make YOLO ideal for a wide range of applications. Similarly, other large-scale dataset annotation pipelines include Google's reCaptcha [46] and Amazon's Mechanical Turk [20, 27]. The issue with these crowdsourced or outsourced solutions to data annotations is that their labeling tasks do not leverage domain or expert specific knowledge.

2.5 UAV Datasets

Aerial datasets can be built with multi-spectral imaging to identify diseases and stress in plants. However, there are many challenges associated with appropriate disease and stress identification, particularly with making accurate labels. Plants are infested with hundreds of pathogens in the field, and most of them can exhibit similar if not identical symptoms. Differentiating between them is no easy task and manual identification by experts is necessary for appropriate disease detection. This is already a challenging task in agricultural activities; making manual identifications based on recorded aerial data implements a new set of challenges. Labeling aerial datasets causes new forms of bias and optical illusions which leads to error. Direct labelling also involves intensive labor with an associated economic cost. Low-resolution imagery is another major challenge, with severe impact on segmentation, classification, and image processing fidelity. This section presents background work in developing aerial datasets and deep learning models for agricultural use.

2.5.1 Data Processing

After building aerial datasets, the UAV images are processed using various tools. For example, vegetation indices are spectral imaging transforms for various image bands that are designed to enhance vegetation properties in images. They are often used by researchers for data processing as they make images easier to read. In deep learning, these types of processing tools can support feature and pattern identification as well as be used for various statistical tools such as K-means clustering and regressions. The choice of data processing method depends upon the purpose of the study; as such, this work is independent of the desired property to be studied.

One of the most common methods of analyzing UAV imagery is regression analysis. This takes places after recorded images are rectified and each imaging channel has been extracted. A regression model is used on this data to investigate the spectral characteristics of the parameters; there are different analysis models to use (linear, non-linear, multiple regression analysis). It is important to cross validate the regression model after analysis. This is because, the model will predict that observations will be the same in the future but in reality that may not be the case [36]. Thus, the data is split into two portions with one set used to form a regression analysis model, and the other is used as a future observation set to fit into the model [33].

Another essential tool for analyzing aerial images in agricultural research are vegetation indices. These indices are used to numerically represent the multi-spectral relationships that are reflected and captured from plant biomasses. Many vegetation indices are practically used to assess plant health such as Normalized Difference Vegetation Index (NDVI) [25], Optimized Soil Adjusted Vegetation Index (OSAVI) [24], and the Crop Water Stress Index (CSWI) [22].

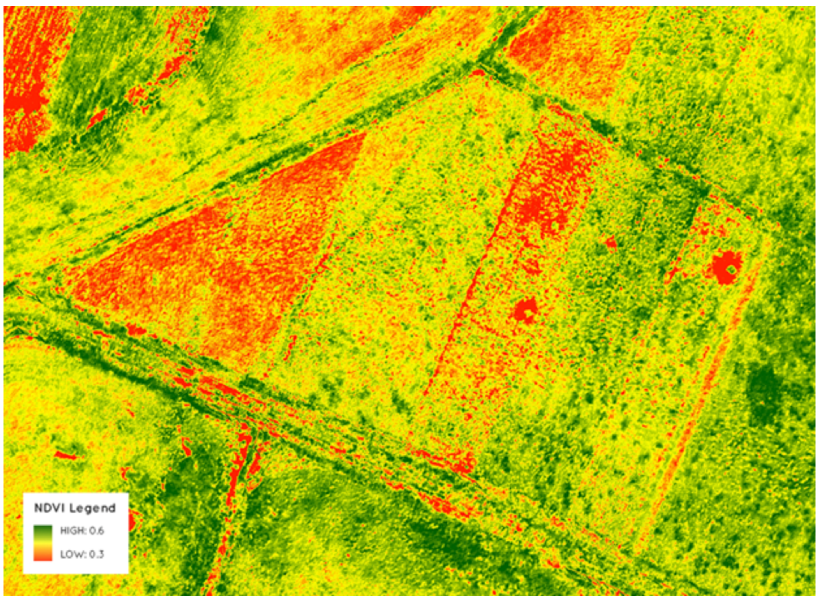


Figure 2.12: Sample satellite NDVI image of a farm. This NDVI scale goes from 0 to 1, where 1 is health and 0 is unhealthy. Green indicates more plentiful biomass and health while the redder areas indicate more water induced stress with less biomass. [39].

Physiological properties such as water content, biochemical composition, nutrient status, biomass content, and diseased tissues are radiantly reflected by plant and captured by these vegetation indices. NDVI is the most common of these techniques to detect diseases, specializing in the red, red-edge, and NIR wavelengths [24]. Analyzing and comparing vegetation indices between diseased and healthy crops is a widely used method for monitoring crop health. These indices offer a simple, easy, and reliable approach for disease identification and monitoring in agriculture [33].

2.5.2 Deep Learning Models

Several deep learning strategies already exist that utilize aerial based imagery. As discussed by Neupane et al [33], many famous large CNNs (VGG-16, GoogLeNet, ResNet, etc.) have been applied successfully for aerial crop health monitoring. The performance of deep learning architectures is influenced by several factors, including the number of training images, minibatch sizes, weight adjustments, and bias learning rates. Generally, deep learning models outperform traditional machine learning methods like Support Vector Machine (SVM) and random forests. For example, CNNs have a correct prediction rate of 1–4% higher than SVMs and 6% higher than random forests. However, CNN models can be 18% less accurate compared to predictions based on Root Mean Square Error (RMSE), according to Song et al [41].

Other studies that highlight varying deep learning performance models such as Too et al. [45] found that DenseNets demonstrated higher accuracy without overfitting, outperforming VGG 16, Inception V4, and ResNet. In contrast, AlexNet showed higher accuracy than SqueezeNet in classifying tomato diseases, while AlexNet and VGG-16 exhibited similar accuracy in the same task. Mohanty, Hughes, and Salathé [30] achieved a 99.35% accuracy rate in plant disease classification using AlexNet and GoogLeNet, although these models performed poorly on different image sets.

As mentioned previously, CNNs are extensively used for identifying and classifying plant diseases from images, aiding in pest control, cropping activities, and yield prediction [49]. These models have have simplified the process to the point for some growers, who can take a picture in the field and upload it to specific software for disease identification. Unlike traditional methods, CNNs eliminate the need for complex feature engineering by identifying important features during dataset training.

However, each deep learning architecture has its pros and cons. As the number of network layers increases, performance degradation can occur, leading to reduced accuracy. Training deep networks is also time consuming due to the large number of images required. Furthermore, deep networks often face internal covariant shifts, which disrupt input data and the training process. Modern techniques such as skip connections, layer-wise training, transfer learning, initialization strategies, and batch normalization are employed to mitigate these challenges [26].

Chapter 3

Methodology

This chapter describes the evolution of a hardware system designed to implement our high-level goal of leveraging ground annotations to automatically label gathered aerial imagery. Overall design goals included the use of commodity sensors and re-use of modular code packages within Linux and ROS2.

3.1 Charlotte & The Sensor Payload

We are utilizing a DJI Matrice 300 RTK quad-copter drone to act as our heavy lift aerial robot. This drone, named Charlotte, was chosen due to prior experience with DJI, its price tag, and its lifting to flight time capabilities. One of the key values of this project, is for our payload package to be removable with stand alone capabilities so that we were not restricted to a single heavy lift robot. For this reason, we made the design choice to not use any of Charlotte's odometry sensors and to instead integrate our own inertial sensors. The M300 line is equipped with real-time kinematic (RTK) functionalities, however, without purchasing the extra proprietary DJI ground station it would be impossible to enable RTK directly onto Charlotte. Charlotte merely only carries our payload while supplying power to our power distribution board via a 24V XT30 port from the OSDK expansion module that we purchased separately. We also utilized DJI-built waypoint missions to fly in familiar spaces as custom autonomous navigation methods were one of our unsolved challenges in this project. Piloting Charlotte within legal bounds was another challenge that will discussed in another section.

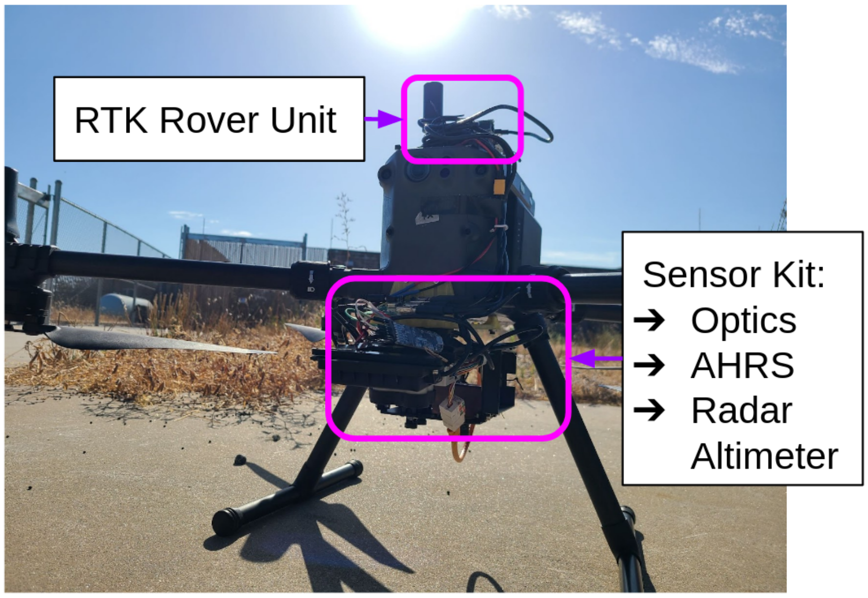


Figure 3.1: Image of Charlotte the UAV with bounding boxes over the sections of the sensor payload. The lower half holds the three camera systems, the attitude heading reference system (AHRS) IMU, and the radio altimeter.[1]

Our sensor payload slots underneath Charlotte's belly with the RTK and UHF radio modules placed atop of Charlotte's head. A mechanical locking mechanism holds the payload in place and allows for easy attachment and removal. The body of the

[1]Figure developed by Morgan Masters.

sensor payload was strategically 3D printed to maximize space and sensor visibility while not interfering the Charlotte's collision detection sensors. Below is a list of our onboard sensors and figure 3.2 illustrates their physical connections.

1. NVIDIA Jetson Nano
2. Micasense RedEdge Mx
3. FLIR Boson 640 Thermal Camera
4. 2 RGB (one without an infrared filter) MIPI cameras (later replaced for a FLIR Blackfly)
5. Ainsein Radio Altimeter
6. Ublox ZED-F9P RTK-enabled GNSS module
7. Lord Microstrain AHRS IMU.
8. External USB Hub
9. RFD900 UHF Radio Modem

3.1.1 The Jetson Nano

The onboard computer, is the Jetson Nano, a contributor to many and more of the projects challenges over the years. We initially chose the Jetson Nano due to it's price tag and its intended function to be used in embedded field robotics and AI applications. The Nano seemed like the optimal pick in the beginning stages of the project, fitting all of our projects specifications and demands. However, as the project

[1]Figure developed by Morgan Masters.

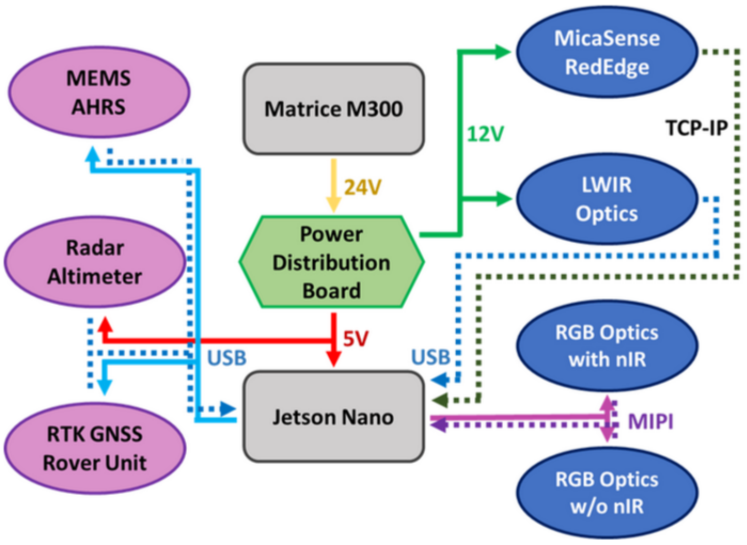


Figure 3.2: Illustration of the system sensors and their physical connections[1].

grew the hardware demands increased; more sensors were integrated which demanded more power and CPU load in order to operate. Table 3.1 lists these specifications. The challenges that the Nano presents tie directly to the restraints given by the memory, CPU, and video encoding/decoding. As such, details pertaining to these challenges will further described in later sections as they directly correspond to sensor or user based functions.

3.1.2 Robot Operating System

Robot Operating System (ROS)[2], is an open source middle-ware robotic event-service framework. It is an industry standard tool that we use to interact with our sensors and record the data that they publish. In general, ROS consists of code and tools

Jetson Specifications	
GPU	NVIDIA Maxwell architecture with 128 NVIDIA CUDA
CPU	Quad-core ARM Cortex-A57 MPCore processor
Memory	4 GB 64-bit LPDDR4, 1600MHz 25.6 GB/s
Storage	16 GB eMMC 5.1
Video Decoding	500MP/sec, 1x 4K @ 60 (HEVC), 2x 4K @ 30 (HEVC), 4x 1080p @ 60 (HEVC), 8x 1080p @ 30 (HEVC), 9x 720p @ 60 (HEVC)
Video Encoding	250MP/sec, 1x 4K @ 30 (HEVC), 2x 1080p @ 60 (HEVC), 4x 1080p @ 30 (HEVC), 4x 720p @ 60 (HEVC), 9x 720p @ 30 (HEVC)

Table 3.1: The Jetson Nano hardware specifications [10].

that help a given project's code run and do necessary tasks including the infrastructure for running it. It is a loosely coupled system where every process, called a node, should be responsible for one task. Nodes communicate with each other using messages passing through logical channels known as topics. Nodes can transmit or collect data from other nodes in a process known as the subscriber/publisher model [44]. Figure 3.3 gives an example of what this process looks like.

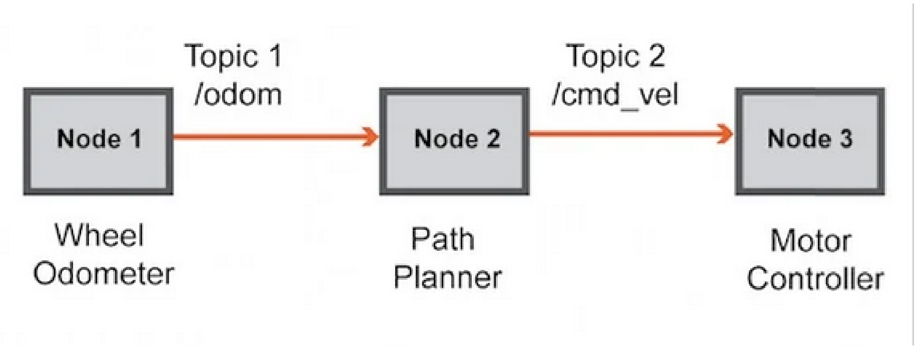


Figure 3.3: Illustration example of ROS nodes and topics. **Node 1** is named Wheel Odometer and its publishes the topic '/odom'. **Node 2** is named Path Planner and it is subscribed to the topic '/odom'. It manipulates the data in some way and publishes the topic '/cmd_vel'. **Node 3**, Motor Controller, is subscribed to '/cmd_vel' and conducts robotic operations based on the messages it receives [44].

ROS2 uses much of the same core components, tools, and libraries as ROS1. It also includes several new features and improvements such as a communication stack with real-time data distribution service (DDS) protocol. DDS acts as a middle-ware for inter-node communication and also utilized quality-of-service (QoS) profiles to provide real time communication, scalability, security, and performance enhancements [17]. ROS1/ROS2 bridges exist in open source communities allowing for some level of interchangeability.

[2]For more information on documentation please reference: https://docs.ros.org/en/iron/index.html

ROS2 Iron Irwin is, currently, the newest version, compatible with Ubuntu 22.04. The Jetson Nano is only officially supported by NVidia for an Ubuntu version no greater than 20.04, which does not permit the use of binary packages pre-built for ROS2 Iron. Thus, the Nano was adapted to adopt a custom headless Ubuntu 22.04 image so that we could take advantage of the newest and most maintained version of ROS2 via prebuilt binaries. Functionally, this modification used the kernel-space from final image released by NVidia with a 22.04 user-space. This design was adopted in order to preserve access to NVidia-proprietary functions such as hardware encoding and GPU functions. Compared to ROS1, ROS2 lags behind with package support, even though its first version was released in 2017. This has required us to develop many custom ROS2 based functions and packages for our sensors. Having a virtual machine that had an installation of ROS1 was also a necessity to use essential packages such as Kalibr, which will be discussed in a later section. ROS1 has significantly more available software packages than ROS2 and for the purposes of this project, it is necessary to use both in order to fully utilize working open source software.

Many of the sensors we chose have ROS2 packages already built and are open sourced on GitHub. Changes were made to each package so that integration could fit with our specification demands. Others sensors required us to build our own custom ROS2 packages from the ground up for integration. Using ROS2 functions, we can 'ros2 launch' our sensors, activating them, allowing them to publish data openly. We can then 'ros2 record' sensor topics to save that data into storage. All of this is done on the Jetson Nano using a remote ssh connection where the Nano hosts an access point using *hostapd*. A launching and recording pipeline have been built that allows for sensor activation and recording of all our desired sensors and topic in two simple commands. This scripting is at the core of our in field operations and allows us to function swiftly to capture data consistently. However, publishing and recording that data is demanding for the Nano and the hardware struggles to keep up. Data rate and data loss have been major challenges and we have attempted to implement strategies to strengthen performance and preserve the data. Storage devices like a NVME SSD were installed to support this data loss, however, compatibility and un-mounting issues prevented this from being a reliable fix. The Nano also sometimes crashes due to an over-demand of power or CPU usage. This would typically occur after launching the sensors where the demand of all the sensors booting up overwhelmed the Nano. Similarly, recording topics often necessitates a majority of the CPU load causing more Nano based crashes and data loss. These issues are still, for the most part, unresolved and without adjusting the hardware to meet the new specification demand it will continue to perform at an undesirable rate.

3.1.3 The Flir Boson

The Flir Boson[3] was our pick for a longwave infrared camera. Due to export control restrictions we were legally restricted to use a Boson that had a frame rate of no more than 9 Hz. This thermal camera does critical work in detecting the reflected temperature signatures off of stressed plants. This is a staple tool for our payload and serves a critical purpose in our ultimate goal in detecting plant diseases.

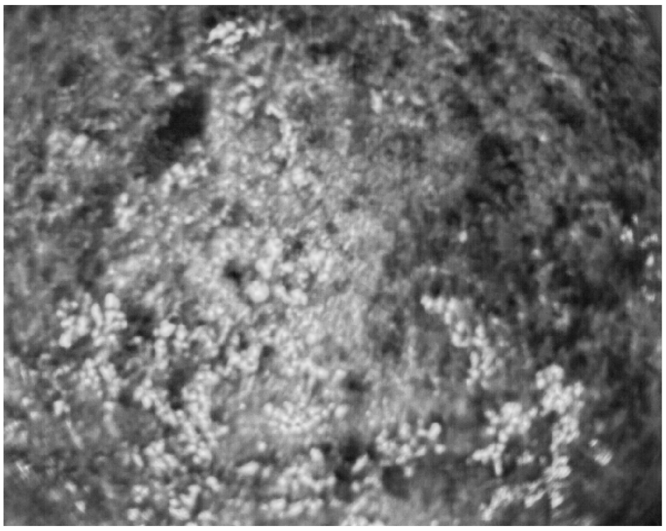


Figure 3.4: Sample data of the Flir Boson from a sample flight meant to look for gopher holes in an open field. The Boson is a looking downward and the heat signatures are clearly visible with black being hotter and white cooler. This image is distorted and unrectified as evidenced by the fisheye style of distortion around the corners and edges.

While there is much utility with the Boson, it has also presented many challenged for us. One such challenge is the electromagnetic interference that is emitted due to its pixel clock. This interference has proved to be a potent GPS jammer, preventing Charlotte's take off in some high obstacle areas, and causing some noise for the RTK system. To counteract this challenge we have implemented a Faraday cage around the Boson by enveloping it in a copper shielding box, grounding it to one of the Nano's exterior ground pins. This has shown to be a somewhat reliable fix for the EM interference. The Boson has also been the most unreliable of our cameras. Upon startup of its node, the Boson demands more current from the Nano's USB 3.0 port. Unpredictably, this can draw too much from the Nano, crashing it. Another issue with the Boson is its

[3] usb_cam ros2 package was used for Boson usage: https://github.com/ros-drivers/usb_cam

uplink port which stops transmitting data when the cable, at the port, is bent close to 90 degrees. This has happened with two different sensor boards and it requires manual cable adjustment in order to regain function.

3.1.4 The Micasense Red Edge

The RedEdge Mx has 5 channels dedicated to each desired spectral band (red, green, blue, red edge, shortwave infrared (near IR)). We have chosen to use the RedEdge as our ground truth camera due to its common use in industry. The intention with this is to compare our ultimate final results, plant stress and disease detection using the RGBs and thermal, against the RedEdge's capabilities to do the same. For the scope of this thesis, the RedEdge was was integrated via a custom ROS2 package and recorded data along with the other sensors. No post processing and analysis was done with the RedEdge.

3.1.5 RGB Cameras

At the forefront of this work, we are using basic MIPI RGB cameras with M12 lenses. The RGB cameras connect to the Nano's CSI ports via ribbon cables. We have integrated them with a custom version of the ROS2 package, 'gscam'[4].

These cameras rely on the Nano's image signal processor, an NVidia hardware component that only has minimal documentation available without NDA. To preserve limited CPU resources, we captured using the NV12 format native to the camera sensors.

[4]For more information on documentation please reference: https://github.com/ros-drivers/gscam

Even with this simplification, we are still have unpredictable and unstable data sampling rates, fluctuating between 3-8Hz and took nearly 70% of the Nano's CPU usage. This will be discussed more in a section dedicated to timing, but the key take away is that the Nano is forced to do this image processing instead of it being offloaded to the camera's hardware (alternative sensors do this like the Blackfly). This, along with the general quality of the RGB cameras, has been a tremendous challenge for us, especially in calibration and sensor fusion.

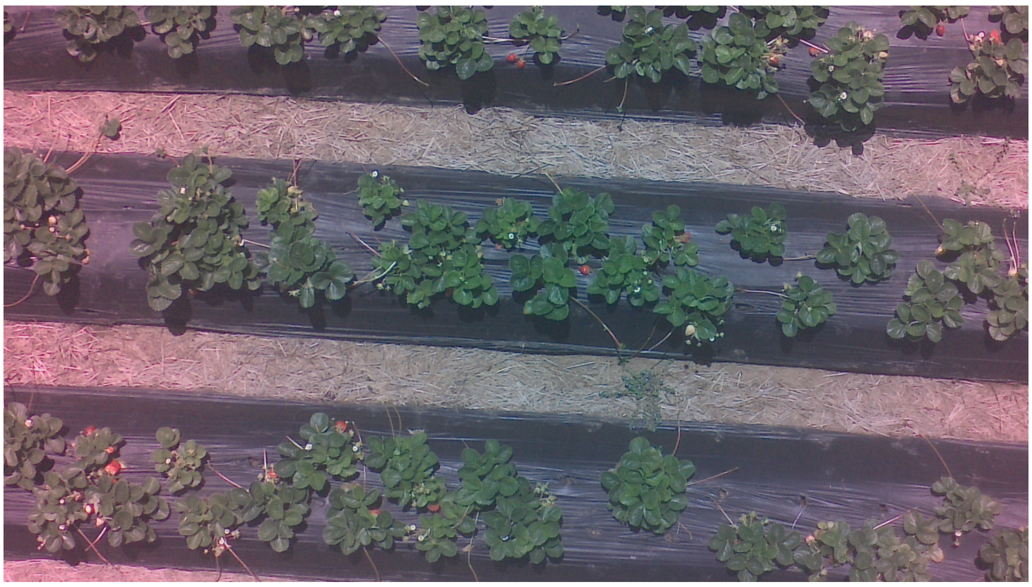


Figure 3.5: Sample data from the no-IR RGB MIPI camera after post-processing. This means that this image was processed to go from NV12 back to RGB and rectification to correct for geometric errors. The red color around the sides indicate that these cameras also require color based calibrations.

Another issue we encountered with these cameras was a electromagnetic based interference occurring on the ribbon cables. Our working hypothesis as to cause relates to cable routing between the Nano's heat sink and the Boson's copper shielding. With the thin nature of the the cables, the current hypothesis is that capacitors inside the cable were affected by the background EM interference in that specific location. This caused the Nano to crash upon attempting to process data coming from the cameras. This was solved by simply moving the ribbon cables to be adjacent to the copper shielding.

With all of these issues surrounding the RGB cameras (more will be mentioned in later sections), we made the decision to retire the pair of RGB MIPI cameras in favor of a FLIR Blackfly GigEVision-based camera. In comparison to the RGB cameras, this camera only takes less than 10% of our CPU load, encoding in mono8, Bayer pattern, or RGB natively. We are able to stably record at 5Hz, and obtain a much higher perceptual quality of an image. With the late timing of this change, only one Blackfly was added with a second without an IR filter being added later during the project's revision stage.

3.1.6 RTK

Real Time Kinematic (RTK) is technique that improves upon the accuracy of a GNSS receiver [9]. It has the potential to have centimeter accuracy with its real time precision. This works by having two receivers, with one stationary and the other placed upon the moving vehicle. The stationary receiver works as a base station, sending positional corrections to the moving receiver. The vehicle receiver uses these corrections to to obtain centimeter level precision.

In our operation[5], we use a Ublox ZED-F9P RTK module accompanied with a UHF radio receiver. The base station send corrections to the onboard payload's receiver

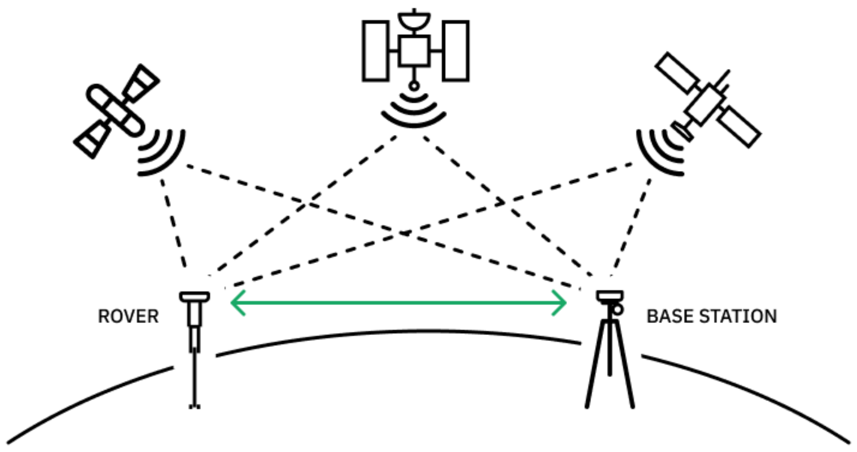


Figure 3.6: Diagram of RTK triangulation between satellites, base stations, and rover receiver. The rover receiver communicates with the base station receiver to make positional corrections. All of this while both receivers communicate with satellites [32].

via the 900MHz UHF signal and obtains no worse than a 3 cm error with a RTK fix. Without the UHF signal, the RTK system reverts to a lower-quality position fix. We average about 75% RTK fix while conducting flight missions. A vast majority of these RTK outages are due to physical obstacles in the flight vicinity which can cause some interference. The range between base station and onboard receiver has not been a cause for outages as our flight area is relatively small compared to RFD900's maximum range.

[5]For the ros2 package used please reference: https://github.com/KumarRobotics/ublox

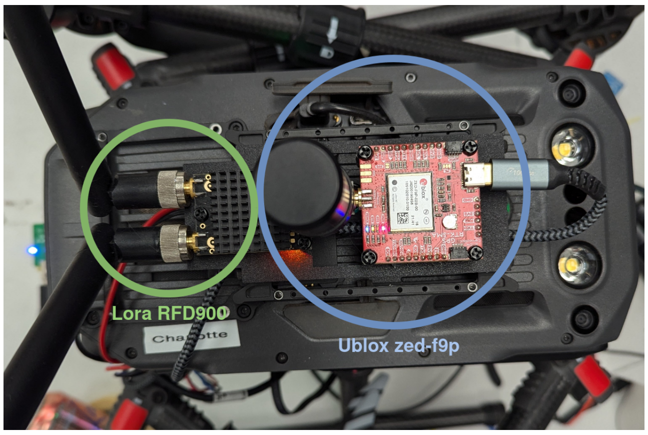


Figure 3.7: Image of “Charlotte’s Hat”, the separated component of the payload where the RTK (in blue) and RFD900 (in green) modules are located.

3.1.7 Sensor Timing

Each sensor, via ROS2, publishes message which are timed stamped when data arrive based on the system time. Each sensor publishes at a different rate, thus having different time stamps per message. This means that in order to have consistent synchronous sampling between our sensors, we have to adjust sensor messages to sample at the rate of the slowest publishing sensor. For us, this slowest publishing sensor is our Ublox ZED-F9P. So, we read the RTK message’s time and pair that data with the other sensor messages that is closest to that time. While this helps out somewhat with our error, it is not highly accurate and we are still left with some minor error due to sub-second timing difference.

The Microstrain IMU is one of these sensors which samples data in an unusual, trimodal fashion caused by a lack of timing guarantees within the USB bulk transport specification. This effect can be better seen in figure 3.16 that illustrates the IMU message timing disparities. This sort of behavior is not ideal as it leads to timing issues with inconsistent sampling rate despite our set sampling parameter to 10 ms. An ideal behavior would be a uni-modal sampling rate where all of the sampling happens around one rate. Other IMU parameters were tweaked to support this issue, such as eliminating the IMU's timing flag, however the Microstrain simply does not have the quality to achieve this based on the fundamental connectivity issue. While the Microstrain integration manual does not recommend the use of USB to read IMU data, there was insufficient time to implement an alternative interface.

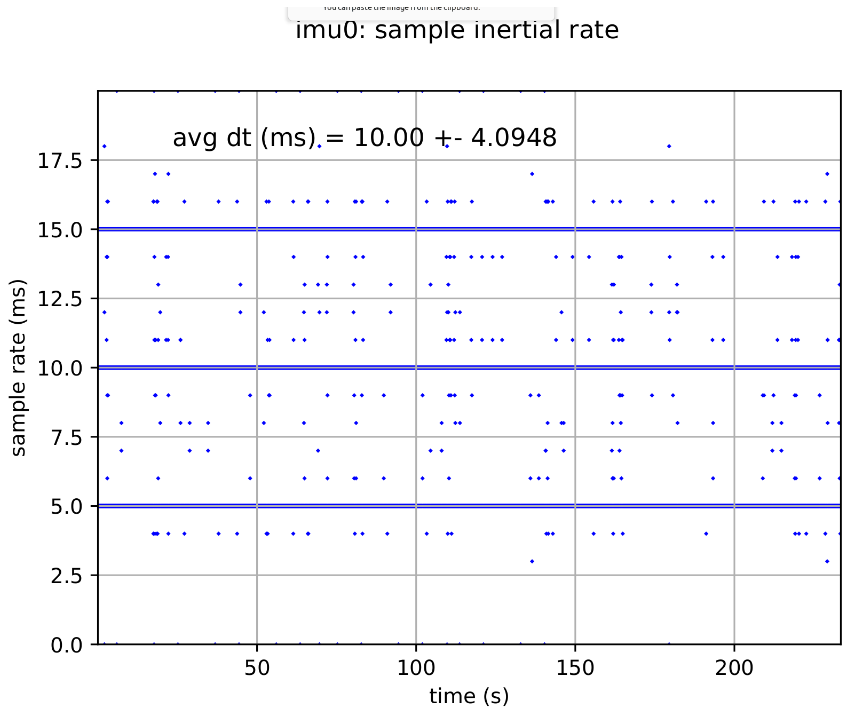


Figure 3.8: IMU sampling rate during a visual inertial calibration. The tri-modal behavior introduces IMU based instabilities that causes timing issues in our system.

Another sensor that was led to significant timing difficulties was our RGB MIPI cameras. As mentioned, these cameras did not encode the images via camera hardware and instead relied on the Nano's software to do it. Not only did this require a massive amount of CPU usage, it also sampled images at fluctuating rates. Simply put the Nano's hardware is not strong enough to run everything and have consistent sampling and timestamps. This led to inconsistencies between both cameras leading to spatial inconsistencies. The camera quality also attributed to timing deviations during calibration but this will be described more in a later section.

3.2 RTK Base Station

In order to have RTK enabled GNSS, we need to have a base station that communicates with the sensor payload to obtain positional corrections. To do this we have made our own field ready base station. To do this, we have employed a 5 gallon bucket with a lid that houses all our hardware. An external power supply powers the electronics inside. Not needing an outlet allows to deploy anywhere in the field and keep Charlotte and the base station in close proximity. Inside is a Ublox ZED-F9P RTK and a UHF module that are velcroed and ziptied to the inner walls of the bucket. Protruding from the top of the lid is our RTK antenna which supplies us with a range long enough to cover the entire 30-acre range of our test farm field.

3.3 The Handheld Clicker

Our final field component is our handheld RTK enabled GNSS point saver or “clicker”. The clicker drops “pins”, an annotated region with its precise position to es-

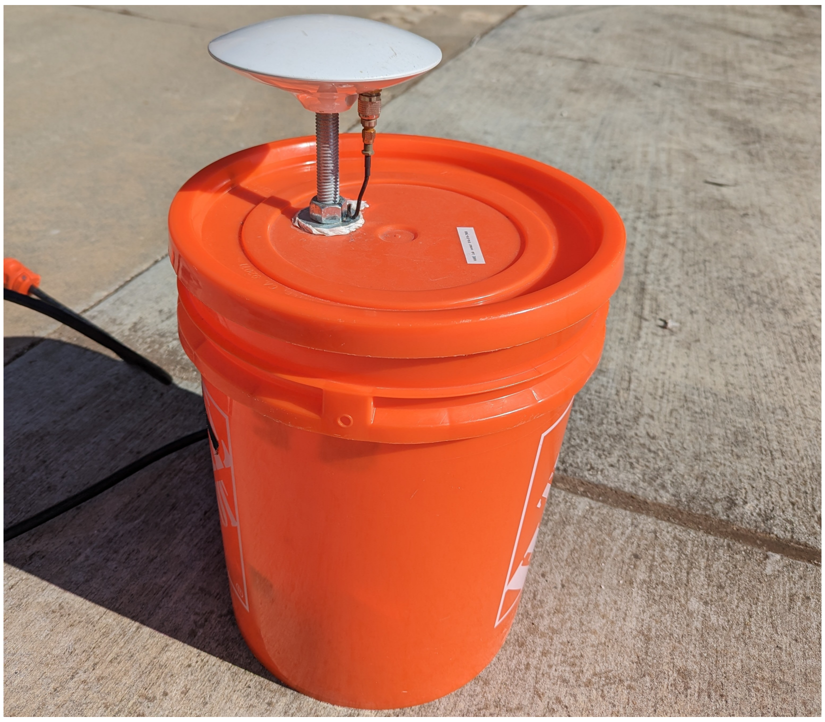


Figure 3.9: Image of our RTK base station. The lid remains removable even with the RTK antenna protruding from the top.

tablish a custom spatial dataset that is later associated with the recorded imagery from the environment. These pins are pointwise markers of the presence of a feature of interest, which can be used as seeds for semantic labels in our multispectral imagery. They contain metadata that makes the pin's information have human readable descriptions or machine learning objectives that users can arbitrarily define. This means that pins can be used for any arbitrary, discrete, and finite set of research targets to be encoded for experimental deep CNN training.

The clicker is built in a similar way as Charlotte's Hat. A UHF and a Ublox ZED-F9P RTK module are used in unison to establish a connection to the base station

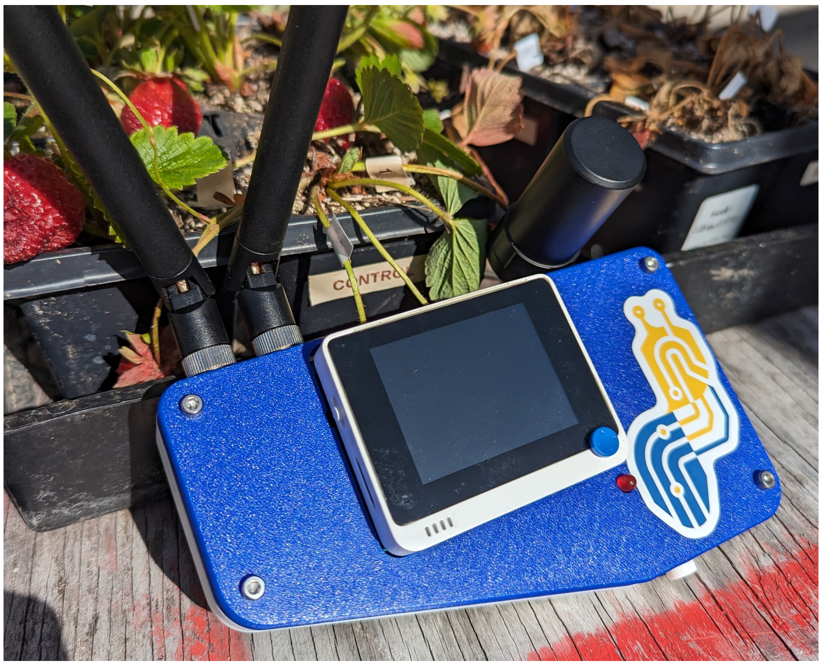


Figure 3.10: Image of the Handheld Clicker. The two antennas on the left are the UHF antennas and the one the right is the RTK antenna. The screen in the middle is the Wio terminal.

and receive positional corrections. A Wio terminal[1] is connected to our receivers and acts as the user interface to save desired pins with corresponding labels. A rechargeable battery is used to power the clicker allowing for prolonged usage in the field.

3.4 BirdsEye

BirdsEye, the novel contribution of this work, is an offline post processing pipeline that takes our recorded data from field operations and associates a relationship between the 3D physical space from pins to the 2D imaging space of the video frames to generate rapidly labeled datasets. Taking the recorded ROS2 bags from field op-

[1]`https://www.seeedstudio.com/Wio-Terminal-p-4509.html`

erations, BirdsEye begins its procedure by pairing together sensor data by their time stamps, converting any custom formats into standard ones, such as NV12 to RGB, and conducting image rectification. BirdsEye then undergoes a series of rotations and translations upon the sensor data to align coordinate reference frames to each other. Figure 3.11 illustrates this concept. Charlotte, the sensors, and the world all have their own reference frames and axis representations and in order to operate mathematical relationships between them, we must make these rotations so our coordinate systems match up. Then, through a series of more rotations, BirdsEye projects the 2D camera imaging frame into 3D space permitting the association between dropped pins into our frame. BirdsEye then re-projects the frame back into 2D pixel space. With this, there is now a pairing between individual imaging frames and real-world features and locations with user defined annotations, or an annotated dataset.

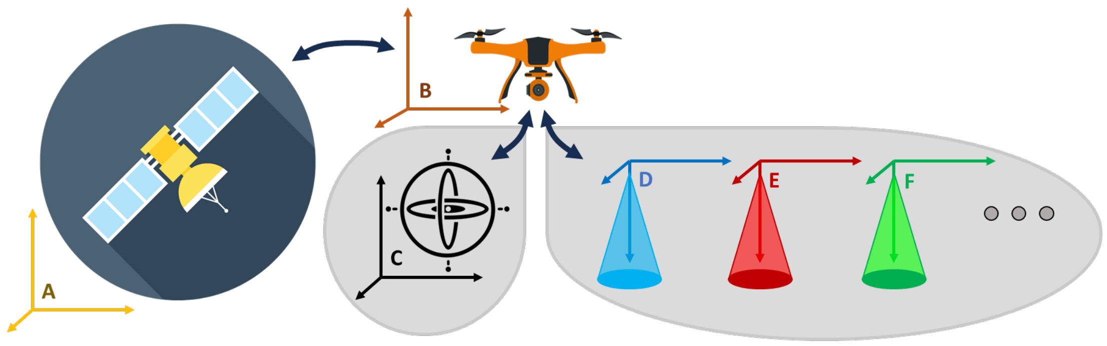


Figure 3.11: There must be accurate estimates of the transforms between: a global reference frame and Charlotte's's control reference frame (coordinate system A to B); the sensor platform reference and relevant flight control systems (coordinate system B to C); each imager's reference frame (coordinate system B to D, E, F, ...)[1].

[1]Figure developed by Morgan Masters.

3.4.1 Spatial Transformations

Describing these spatial rotations are necessary to understand to BirdsEye's core functionalities. The critical 2D to 3D and 3D to 2D rotations are inverses and the math associated to their calculations reflect this inverse property. For this reason I will discuss the math associated with the transformation from the 3D to 2D perspective which functions as BirdsEye's backprojection/reprojection where pins are brought into pixel space. The conversion of a 3D point on world coordinates to the 2D point on the screen can be described as the following:

$$\begin{bmatrix} x_{vp} \\ y_{vp} \\ z_{vp} \end{bmatrix} = M_{vp} M_{per} M_{cam} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

where the viewpoint coordinates x_{vp}, y_{vp}, z_{vp} is equal to the matrix multiplication of the view point matrix (M_{vp}), the perspective matrix (M_{per}), the camera matrix (M_{cam}) and a 4D representation of the 3D world frame point $(x_w, y_w, z_w, 1)$ [16]. The function, while not complicated from a glance, can be quite tricky to compute properly. This is primarily attributed to the first three matrices in the matrix multiplication as extracting certain parameters can be a challenge. BirdsEye does not require the viewpoint and perspective matrices as they solely function to make minor positional corrections and perspective alterations. Nevertheless understanding these matrices can support the fundamental foundation to these spatial transformations.

3.4.1.1 Camera Matrix

The camera matrix models the camera's position and orientation ensuring that whatever the camera sees gets projected relative to the camera's perspective. Essentially it rotates the coordinate frame to match the camera frame so that all other operations are carried out in that alignment. In general this alignment is comprised of a rotation matrix times a translation matrix which can be generally expressed as:

$$M_{cam} = \begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -e_x \\ 0 & 1 & 0 & -e_y \\ 0 & 0 & 1 & -e_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The translation step shifts the origin of the camera's coordinate system to match the origin of the world's coordinate system. To do this, the negative position of the camera relative to the world frame must be used in calculation to achieve the correct alignment. To help support mathematical simplicity, the matrices are homogenized to represent four dimensions. This allows the use of dot products for the translation which offers certain advantages. Translation alone is not enough to complete the transform. To completely align the coordinate frames a rotation is applied along the Cartesian (x, y, z) camera axis and a dot product is performed with the point vector [16]. This camera model is extracted for us via Kalibr based calibrations which will be discussed further in a later section.

3.4.1.2 Projection Matrix

There are two fundamental ideas that grasp the concept of projection: “view volume” and “canonical view volume”. View volume encapsulates the volume that the camera can see within the 3D scene. Objects outside this volume, logically, won’t show up in the image. Canonical view volume takes the view volume and transforms it into a cube centering it at the coordinate origin and extending from -1 to 1 along the x, y, z axes.

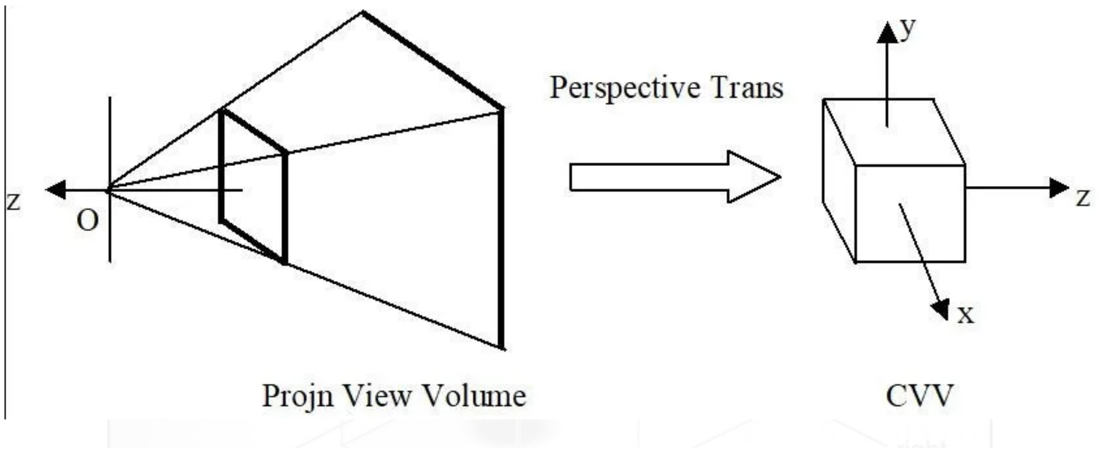


Figure 3.12: Conversion of View Volumes to Canonical View Volume (CVV). The frame from the projection view volume undergoes a perspective based transform to turn what the frame sees into a cubic canonical view volume with aligned coordinates [16].

We can consider projection to be the process that converts the view volume into this canonical view volume where there are two classifications of projections: orthographic and perspective projections. Orthographic projections serves to ensure that objects retain their size in the image. This is done by fitting the object within a rectangular prism-like view volume which ensures that the object’s dimensions match the image dimension. Perspective projections are used to make objects appear smaller when they are farther away in he frame. A prism-like shape is used to do this, where the part corresponding to distant objects has a larger area.

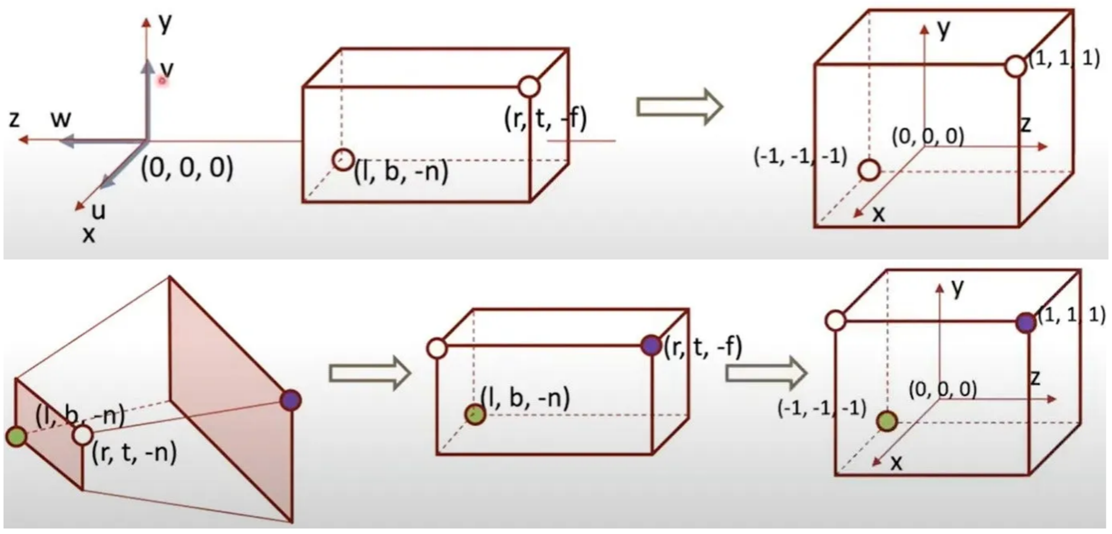


Figure 3.13: **Top:** Illustration of the orthographic projection with two objects labeled with coordinates (l,b,-n) and (r,t,-f). Objects here within the canonical view volume retain their volume without altering their proportions. **Bottom:** Illustration of the perspective projection with two objects labeled with coordinates (l,b,-n), in green, and (r,t,-f), in purple. Objects here with in the canonical view volume that are far away appear smaller compared to objects that are closer as seen with the green and purple points [16].

The perspective matrix used for spatial representations can be calculated by taking the dot product of the orthographic projection matrix and the perspective view volume to orthographic view volume matrix. Using Figure 3.13 as an example above we can represent the perspective matrix as the following:

$$M_{per} = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

The values in this vector represent the point's precise position in the canonical view volume. The transformation from the 3D world into the canonical view volume is therefore complete with adjusted values that hold the point's location information in the cube-like space [16].

3.4.1.3 View Point Matrix

The view point matrix enables us to tell where the a point within the canonical view volume will be positioned on the screen. This is best understood by examining Figure 3.14. In the image, *nx* represents the screen's width and *ny* the height. A transformation about the points within 2D [-1,1] range occurs to fit within the the [-0.5, nx-0.5] and [-0.5, ny-0.5] range. Along the z-axis this transformation fits the point within [0,1]. This depth feature helps determine whether a point is positioned in front of or behind another point.

For this example, we can model the view point matrix as the following:

$$M_{vp} = \begin{bmatrix} \frac{n_x}{2} & 0 & 0 & \frac{n_x-1}{2} \\ 0 & \frac{n_y}{2} & 0 & \frac{n_y-1}{2} \\ 0 & 0 & \frac{1}{2} & \frac{1}{2} \end{bmatrix}$$

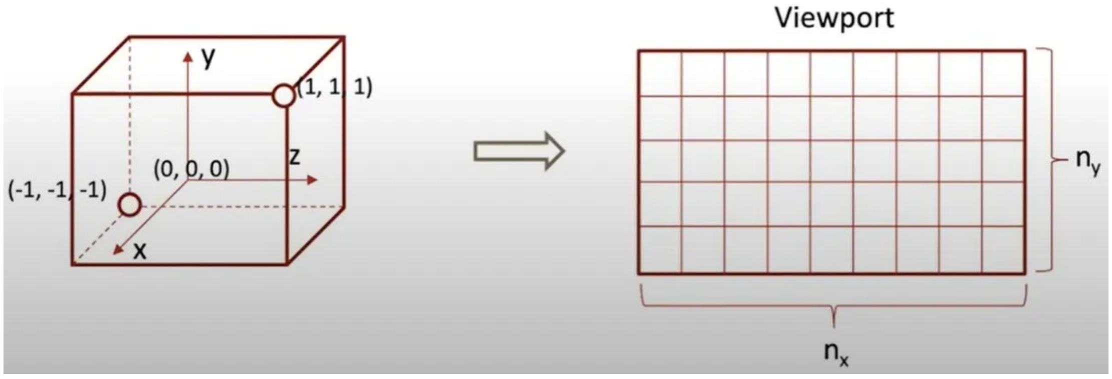


Figure 3.14: Conversion of canonical view volume to image screen coordinates. The volume is flattened with objects being transformed to fit the 2D view [16].

3.4.2 Accuracy

BirdsEye must be very accurate, with near perfect modeling, in order to have properly placed labels over desired features with minimal error after re-projection. There are many variables that affect this error and getting them as accurate as possible has been one of the major hurtles of this work. One of these hurtles is obtaining accurate camera models and camera position to other sensors. Understanding the cameras helps us make mathematical inferences about how the geometric shape of the lens, the focal length, and its overall position affects what we see. This is at the core from going to 2D to 3D space and vice versa. Camera calibration is how we obtain this information, and it has been a notorious challenge for BirdsEye. Inaccurate calibrations, to a precise centimeter level, can cause substantial error in BirdsEye. Not to mention that with our other major sources of error like timing, and our flat Earth assumption BirdsEye can produce results of dramatic disproportions. We assume the Earth is flat, not due to misinformed scientific research but, because it is easy to model and the small scale at which we fly makes the perception of the Earth to be relatively flat. What we lose from this assumption is geographic terrain features such as inclines. Our altimeter does pick up changes in altitude but in the scope of our imaging frames, we are unable to tell the geometry of what we are seeing. Thus BirdsEye has the potential to completely misinterpret pin locations after re-projection if the terrain features are dramatic enough.

This all has resulted in much of BirdsEye work being involved in minimizing these sources of error. Cameras need to be accurately calibrated, timing between sensors need to be tight, and we must have an accurate terrain representation model (such as a Digital Elevation Model or DEM). The bulk of this work has gone into the two former tasks, with developing a working DEM being consider future work. As discussed, we have attempted to solve the sensor timing issues via software, however, we are at a point where only better hardware can improve this error. As for calibration, this will be discussed in the next section.

3.5 Calibration

As previously mentioned, camera and sensor calibration has been one of the toughest challenges of this work. To conduct sensor calibration we use an open source software and industry grade tool known as Kalibr. Kalibr has the ability to produce reports that provides us necessary information of each camera model to a very precise margin using an optimization to estimate camera parameters. The intrinsic matrix, which represents the camera focal length and optical center, and extrinsic matrix, the

BirdsEye Procedure	
Step 1	Flight imaging and positional data are recorded and in field pins are saved via the Clicker.
Step 2	Data is transferred and processed locally where time stamps are aligned and images are undistorted and rectified.
Step 3	BirdsEye spatial rotations occur where pins are associated to the imaging frame.
Step 4	In the images, pins become semantic annotations for our labeled dataset.

Table 3.2: A basic step by step guide on how the BirdsEye pipeline works. First the data and pins are collected during a flight mission. Next the data is extracted from the Nano and is processed locally where image frames are undistorted, rectified, and corresponded to their respective position and orientation. In step three, BirdsEye conducts its spatial rotations using calibration information where pins are placed into the imaging frame. Finally, the labeled dataset is saved wherein frame pins become semantic annotations.

transformation matrix between camera and world coordinates, are examples of some of these calibration outputs. Calibration can also be conducted between different imaging and IMU sensors. Multi-camera calibration is a useful technique to get transnational and rotational information between each camera, along with their intrinsic parameters. Similarly, a visual-inertial calibration can also be conducted between the cameras and the IMU to get a spatial rotation between them. This rotation is exceptionally important to track the movement of the camera frames. To calibrate the cameras, we use an Aprilgrid, an assortment of Apriltags onto a single target. We then record a bag, waving our cameras in front of the target, exciting all of its modes by maximizing the in-frame coverage and making rotations about each axis. Kalibr is only ROS1 compatible and thus we take this bag and input into the software locally, along with other necessary parameters such as the Aprilgrid dimensions.

These calibration outputs must be highly precise in order for it to be used for BirdsEye. Slight deviations in the exact model can lead to bad results and getting good results can be an extreme challenge. Even for simple single camera calibration, Kalibr has the tendency to not converge which does not provide an accurate report. Even if a report is generated, it, often times, is not accurate which leads to more complicated calibrations (multi-camera, or visual inertial) to be even worse. In the following, we present more detail about our calibration features, the issues that occurred, why they occurred, and how we got around them.

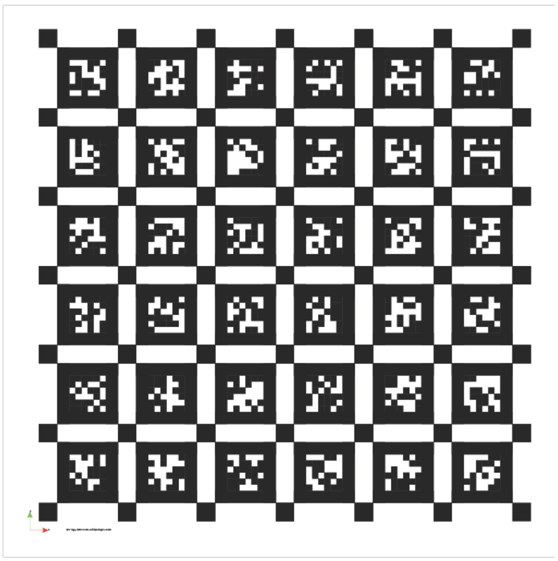


Figure 3.15: Image of the 6x6 AprilTarget used for Kalibr camera calibration. Tag width and tag spacing must be known and placed into a .yaml file for calibration to occur via Kalibr.

3.5.1 Sensor Orientations

We need to have the proper mathematical transforms between each sensor for accurate rotations. This essentially means that we need to know where each sensor is in respect to one another other. For simplicity, the IMU, as the point of inertial origin, serves as our base from origin. BirdsEye's performance heavily relies upon precise translational vectors; these rotations include GPS to IMU, altimeter to IMU, and cameras to IMU. Conducting a calibration via Kalibr is not an option for the other sensors that are not cameras or the IMU. Thus, the only way to find these extrinsic parameters are by using simulated Computer Aided Design (CAD) models of our system and/or using measuring tools. This is not nearly as accurate as Kalibr with the measuring center of each sensor being impossible to pin point from an outside viewing perspective. Out of all of our estimated translational vectors, the GPS to IMU values carry the roughest of the estimates with the large cylindrical RTK antenna providing centimeter level variability to our estimated sensing center.

Aligning sensor coordinate systems is also necessary to ensure proper translations. As we are not using any of Charlotte's built in sensors, the vehicle reference frame is pinned to the IMU. Its coordinate system can be seen in Figure 3.16. Aligning all sensor coordinate system is a tricky process that requires a lot of visualization and testing. ROS Enhancement Protocol (REP) 105 [12] gives a default ROS-based coordinate frame that developers use for shared spatial conventions. This coordinate frame is in East-North-Up (ENU) where the x-axis aligns east, y-axis north, and the z-axis up at the origin of the frame. To accommodate this, a flag was set in the IMU parameters that transforms its coordinate system to fit the ENU system.

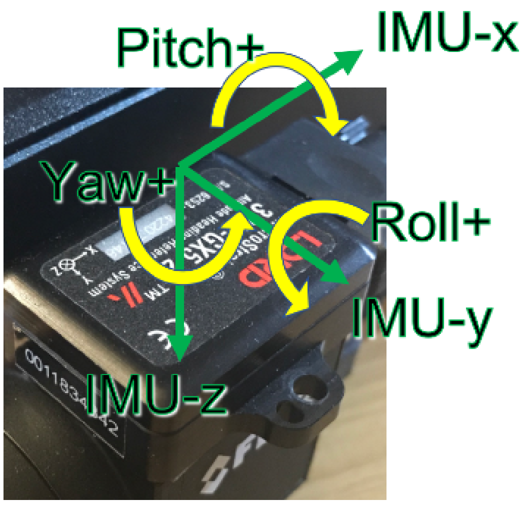


Figure 3.16: The default Lord Microstrain AHRS IMU coordinate axes. The z axis points down, x points North, and y points East. Another term for this coordinate system is North-East-Down (NED).

The REP 105 system, seen in figure 3.17, can be represented by a transformation between four different reference frames: Earth, *map*, *odom*, and *base link*. The *base link* frame or our IMU frame is rigidly attached to Charlotte and the sensor payload. The *odom* frame is a world-fixed frame where Charlotte's pose is continuous, despite drift. It is an accurate short-term local reference that utilizes both IMU and GPS data. The coordinate frame called *map* is a world fixed frame where the z-axis points up. This frame is not continuous, where Charlotte's pose can change in discrete jumps at any time. This frame useful as a long-term global reference but with sensor drift, jumps can occur making it a poor local sensing tool. Finally, the Earth frame allows for the interaction between multiple robots in different *map* frames. Earth frame is a global reference that allows for the *map* frame to initialize its absolute position. For BirdsEye's purposes, we care more about the *odom* reference frame to help us with positional based corrections. We visualise this frame to validate the cohesion between of positional based sensors.

3.5.2 Camera Calibration & Image Rectification

Camera calibration provides us crucial information on the camera model such as the intrinsic and extrinsic matrices. These parameters are essential for spatial rotations (3D to 2D and vice versa). We can express our pinhole camera model as

$$P = K[R|t]$$

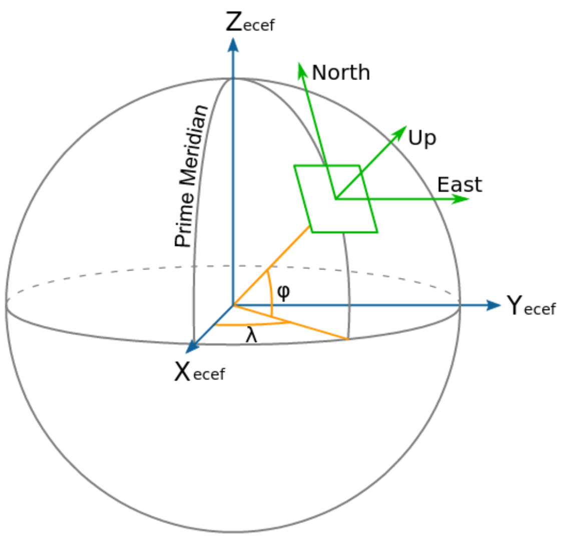


Figure 3.17: A visualization of Earth Centered Earth Fixed with a tangential map frame. In the ENU system, the coordinates wraps around the Earth's curvature. In green we can see the *map* frame following the ENU system [12].

where K is the intrinsic and $[R|t]$ is the extrinsic. We can express K as a matrix representation of the focal length(f_x, fy), the optical center (c_x, c_y), and the camera skew (s):

$$K = \begin{bmatrix} f_x & s & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

The extrinsic is represented as a transform matrix consisting of a rotational component $R_{i,j}$ and a translational $t_{0,1,2}$ component:

$$K[R|t] = \left[\begin{array}{ccc|c} R_{1,1} & R_{1,2} & R_{1,3} & t_0 \\ R_{2,1} & R_{2,2} & R_{2,3} & t_1 \\ R_{3,1} & R_{3,2} & R_{3,3} & t_2 \end{array}\right]$$

Kalibr is able to calculate these matrices for us, as they are unique to each camera, after inputting a ROS1 bag of Aprilgrid camera data. For a single camera calibration, Kalibr reports this intrinsic matrix, along with the reprojection errors. Multi-camera calibrations handle multiple cameras at the same time, providing us with each of their intrinsic parameters and the extrinsics between them. This information gives another point of reference in the spatial transforms (3D to 2D).

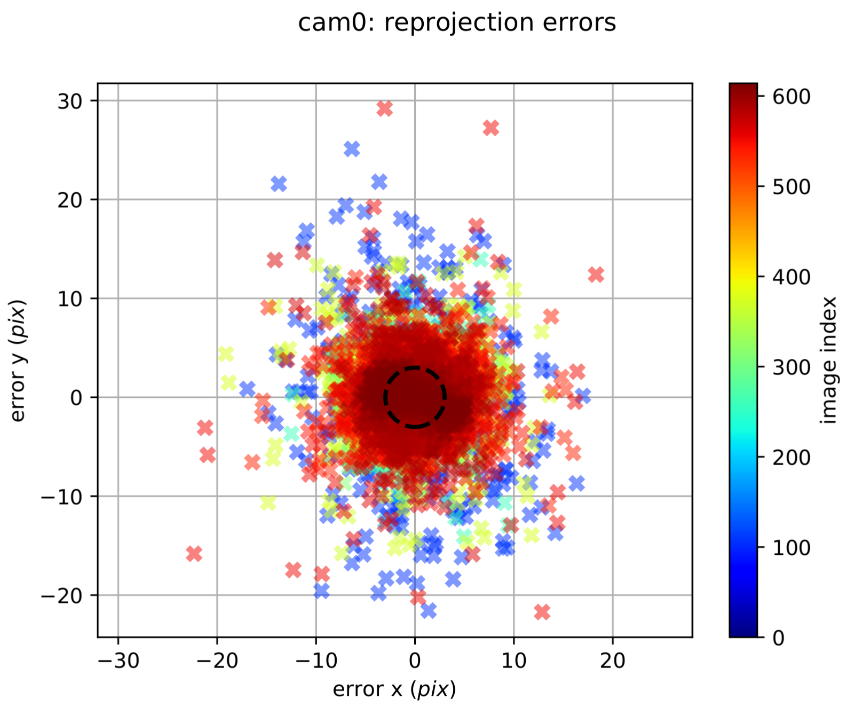


Figure 3.18: Flir Blackfly reprojection error after a calibration. This good result shows that our reprojection errors are well localized in the center of our camera.

We encountered significant problems when trying to calibrate the original MIPI cameras. While we could recover single-camera intrinsics, multi-camera calibration proved to be inaccurate and error prone. This is most certainly due to a MIPI camera timing issue, as variance in timestamp to frames causes the cameras to appear to be in different positions than they actually were. Based on Kalibr reports there is a timing difference between cameras of about 0.5s. To test this further, we conducted another simple test. Playing the ROS time on a screen, we continuously recorded the screen with the displayed time to then play back the recording and compare the time stamp to the time message seen on screen. This result seemed to correspond with Kalibr's 0.5s timing difference and presented a new odd behavior. At some point during this test, the timing difference turned negative where the timestamp was greater than the on screen time. This highly concerning behavior indicates a much larger issue happening at the fundamental level of how ROS is interacting with the Nano's base hardware; recall that several components of the Nano's MIPI interface are not publicly documented. As we could not work around this limitation, we replaced the MIPI cameras with a GigEVision-based camera, the Flir Blackfly. Our visual calibration to extract camera intrinsic was done outside were we could narrow the aperture of the lens to better simulate field conditions. This calibration was successful with our reprojection errors being demonstrated in Figure 3.18. Despite a few outliers our error is localized very well in the center of our lens.

The infrared camera were also a challenge to calibrate. As it only sees thermal data, our first hurdle was to reflect enough heat off of the target so that features could be detected. Placing the target in the sun's radiance, resulted in enough deviation between the black and white patterns of the target, however, there was not enough fidelity to notice the the sharp changes in each square thus Kalibr was not able to detect the target. Our solution was to change the target to be a checkerboard with a plywood backboard, a compatible but not recommended target for Kalibr. Kalibr still had a difficult time seeing the target but with a longer bag, with more exposure time, Kalibr finally yielded an output report.

Image rectification is another transform that we conduct in BridsEye that projects images onto a common image plane. This is necessary because camera lenses and orientations distort the images and they need to be corrected in order to make accurate spatial references to each other. If this is not done properly then pins that appear in the edges of the images can be spatially distorted. Figure 3.19 illustrates this process where we have an average of 1.73 pixels of reprojection error and a 1.68 pixel standard deviation.

3.5.3 Visual Inertial Calibration

A visual inertial (VI) calibration, is a calibration technique between cameras and our inertial measuring unit. The output, in Kalibr, creates a report that details the intrinsic matrices, extrinsic matrices (between cameras and IMU), pose estimation, and an assortment of IMU error characteristics. VI calibration is a complex yet critical process that involves accurately determining and refining the spatial and temporal alignment between the cameras and IMU, allowing us to track changing motion in frame. To

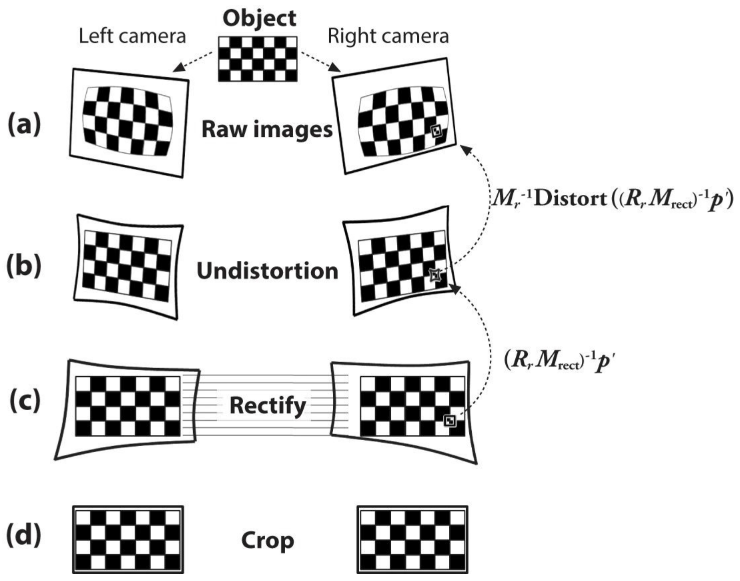

Object
Left camera
Right camera
(a)
Raw images
M_r^{-1}Distort$((R_r M_{\text{rect}})^{-1}\boldsymbol{p}')$
(b)
Undistortion
$(R_r M_{\text{rect}})^{-1}\boldsymbol{p}'$
(c)
Rectify
(d)
Crop

Figure 3.19: The image rectification process. In (a) images are received by cameras with lens distortion as seen by the warped edges. (b) undistorts the images and in turn warps the frame. (c) rectifies the image by adjusting them to share the same axes. Finally in (d) the images are cropped.

ensure peak performance, the cameras need to be recorded at a high quality as Kalibr associates excited IMU modes to the nearest frame with a detected AprilTag. This means that if the images are too blurry and AprilTags are not often detected, there will be a vast time difference between the IMU and camera leading to poor spacial awareness and a bad calibration. Not only did the MIPI cameras have poor timing already, but their quality and high default zoom made it a lot more difficult to calibrate. The calibration report used in this thesis corresponds to the Blackfly. To conduct a VI calibration, we must input two other calibrations as parameters: a multi-camera calibration and an IMU random walk.

The IMU bias random walk is based upon sampling the IMU data over a prolonged period of time and calculating the Allan Variance parameters to determine the noise parameters of a MEMS gyroscope. In other words, it estimates three noise parameters N (angle random walk), K (rate random), and B (bias instability) from a stationary gyroscope [6]. This can be computed mathematically as:

$$\Omega(t) = \Omega_{ideal}(t) + Bias_N(t) + Bias_B(t) + Bias_K(t)$$

Our IMU random walk was taken for 18 hours and produced reports corresponding

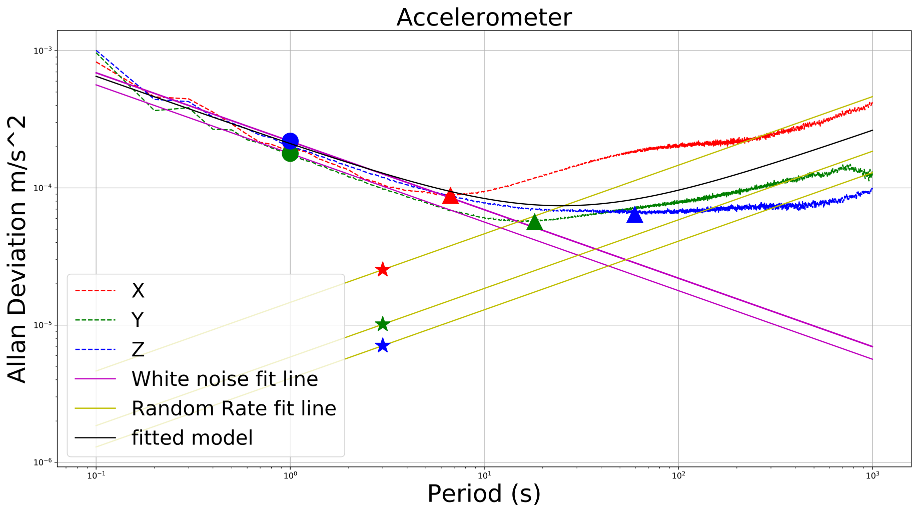


Figure 3.20: Accelerometer Allan Deviation over a logarithmic time scale for a 18 hour recording. The y and z-axes are fairly tight together but the x-axis seems to jump early indicating to a lack of quality with the accelerometer.

to the IMU's acceleration and gyroscope error. Figure 3.20 and 3.21 illustrate these behaviors on a logarithmic scale. At short observation times, the Allan deviation is high due to noise and it begins to decrease as the noise averages out. Over a long period of time, we can see that it begins to increase again suggesting the clock frequency is gradually shifting due to temperature changes or aging degradation of components. The deviation values show that there is quite a bit of error that the IMU experiences. This

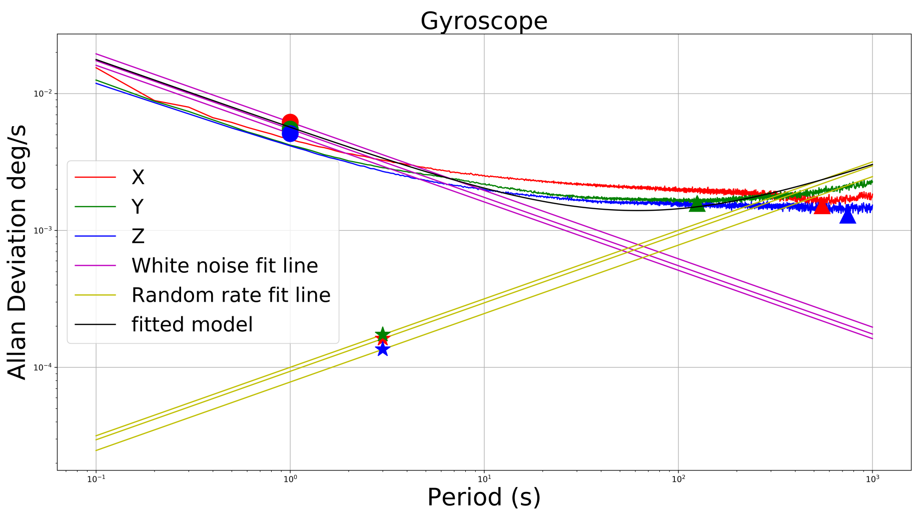


Figure 3.21: Gyroscope Alan Deviation over a logarithmic time scale for a 18 hour recording. The axes deviations align tightly with each other indicating consistent predictable error.

hypothesis is consistent with the strange behavior observed with the accelerometer's x-axis. After a VI calibration this behavior in the biases can be further seen with Figures 3.23 and 3.22. While the gyroscope exhibited a stable behavior, the accelerometer, particularly the x-axis acceleration, produced variable if not unstable characteristics. This could lead to positional inaccuracies in BirdsEye. After the VI calibration it was reported that we have a 2.13 $\frac{m}{s^2}$ IMU error and a standard deviation of 3.1 $\frac{m}{s^2}$.

The multi-camera calibration serves to input the camera model (individual intrinsic matrices and their shared extrinsic matrix). However, as previously mentioned due to the timing issues surround the previous MIPI cameras, multi-camera calibrations produced poor results. To attempt to get around this, we fabricated our own multi-camera calibration by cannibalizing the working components of other calibrations and using the CAD file to estimate the extrinsic parameters. This allowed us to get the

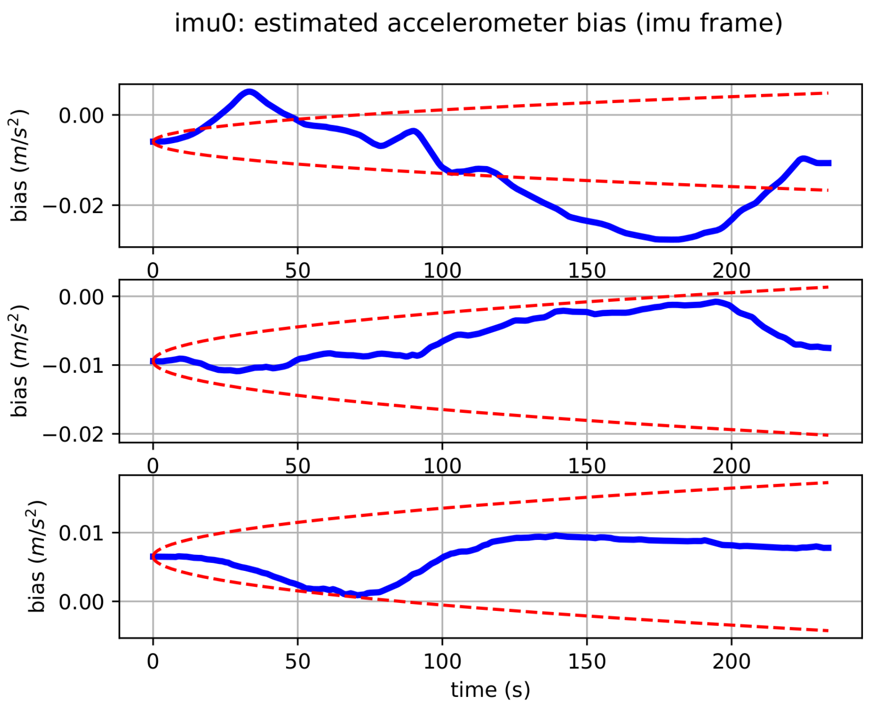


Figure 3.22: The IMU acceleration bias behavior after a VI calibration. From top to bottom these biases correspond to the x, y, then z axes. As seen with the Allan variances, the x-axis bias is volatile and somewhat unstable.

closest to a successful VI calibration but the extrinsic between the cameras and the IMU, was still not accurate enough. The small variances in the intrinsic and extrinsic as well as the timing issues still provided too much error.

Our Blackfly produced great visual calibration results. The only challenge surrounding the Blackfly calibration was obtaining the most depth of field while incurring as little motion blur as possible. To do this we conducted our VI calibration in a controlled indoors environment where the camera f-number was set to the lowest possible to yield the largest aperture size. This allows the camera to intake more light but it sacrifices the depth of field. We did this to extract the camera-IMU extrinsic

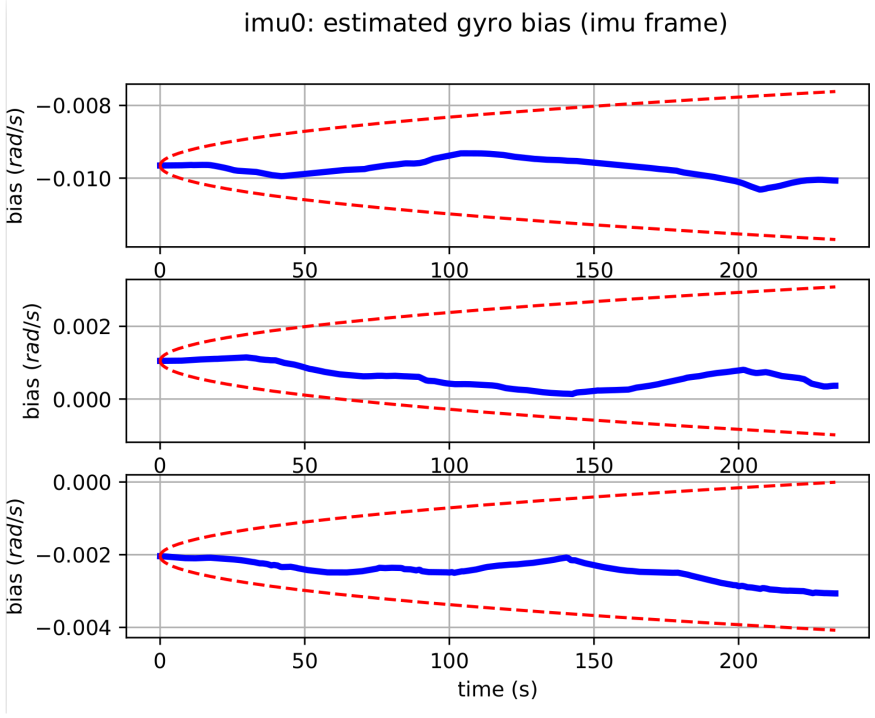


Figure 3.23: The IMU gyro behavior after a VI calibration. From top to bottom these bias correspond to the x, y, then z axes. As seen with the Allan variances, the biases are quite stable.

parameters. For the intrinsic parameters, we conducted a visual calibration outside in practical conditions, narrowing the aperture slightly to increase depth of field. We then merged theses intrinsic and extrinsic parameters in BirdsEye.

3.5.4 State Estimation

State estimation is a necessary component of understanding a vehicles dynamics. Conducing state estimation is typically done for building autonomous navigation systems, however, this level of positional calculations can estimate our odometry and help correct the positions used in BirdsEye. To make these state estimations we employed a ROS2 package known as robot localization. This, via sensor fusion techniques, takes IMU and RTK-enabled GPS points and accurately estimates the vehicle's states. Mathematically, this sensor fusion is calculated through an Extended Kalman Filter (EKF). Kalman filtering, or Linear Quadratic Estimation (LQE), is an algorithm that takes in sensor measurements including statistical noise and other inaccuracies over time then calculates estimates of unknown variables. EKFs are a nonlinear version of Kalman filtering which linearizes a model about estimates of the current mean and covariance. It is an industry staple for nonlinear state estimation.

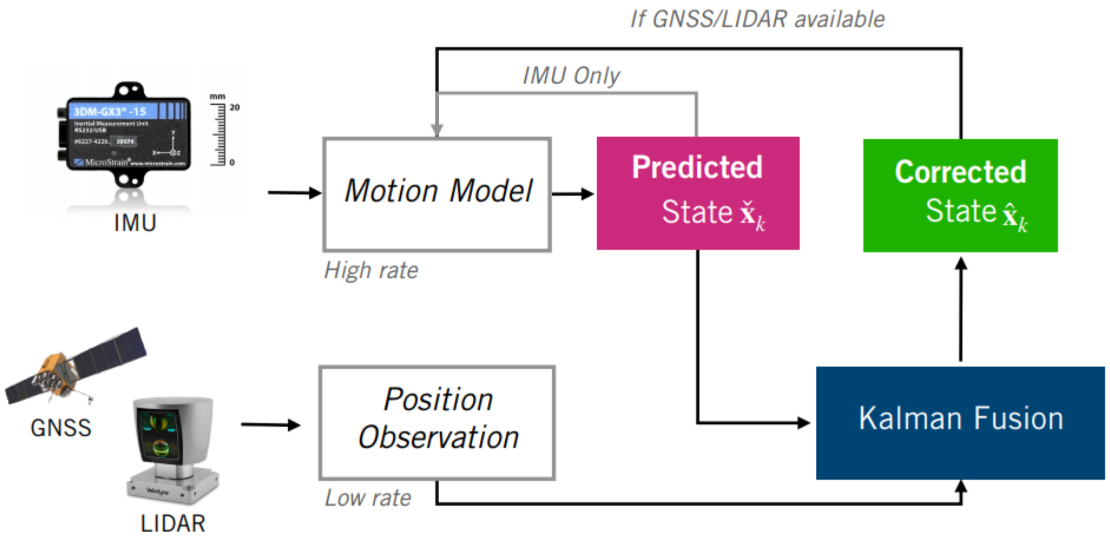


Figure 3.24: State Estimation example framework. IMU and GNNS data points get fused together to create a correction of the vehicles state. These estimates are updated back into the motion model that makes corrections [3].

This is the final piece of the puzzle for BirdsEye to make accurate system level positional corrections and camera frame based predictions. With state estimation, calibrations, and our 3D to 2D rotations, BirdsEye has the mathematical tools it needs to make accurate space-based inferences.

Chapter 4

Results & Experiments

The BirdsEye pipeline is the novel contribution of this work and it has a the potential to improve the field of field robotics in precision agriculture. The system must be meticulously tested and validated in order to ensure the accuracy that is needed to have this desired impact. While more testing in different environments will need to be done in the future, this work focuses on achieving the most fundamental validation test we derived, the "Polygon Flight Test". This test will be described more in detail but essentially it allows us to compare our back projected pins in our image to their ground truths. In this test we should see near perfect overlap between our back projected pins and the ground truths. In our pursuits to achieve these results we have been far from these ideal overlaps primarily due to hardware based issues mentioned in the previous sections. Even with several changes, our back projections did not seen to be adjusting its positions properly having a consistent spatial offset. This can be seen in figure 4.1 where the ground truth points in green do not line up with the back projections in blue.

In this this polygon flight, images were recorded with the older MIPI cameras.

What this data has shown us is that our predicted frame position does not line up well with reality, where we sometimes compute a disparity of over a meter. This is due to having bad positional estimation which we have narrowed the potential causes from things such as timing and bad hardware performance. While we have replaced some hardware, such as the cameras, which have introduced timing issues, we can also improve our positional corrections by implementing independent EKF based sensor fusion in the form of odometry. With this feature, we have updated our polygon flight test to be a “Odometry Enabled Polygon Flight” that incorporates better positional estimation and a higher quality camera. In the following results we should see a significant improvement compared to what is illustrated in Figure 4.1.

4.1 Experiments

The following section outlines the fundamental polygon flight experiment to validate BirdsEye and the onboard sensor payload. Using the systems mentioned in the previous sections, this practical experiment validates the the ability to be used in the field in an accurate and informative manner. To conduct this experiment, Charlotte must be flown by a licensed FAA part 107 remote drone pilot that knows and understands the laws and legislation associated with flying a drone. This is a requirement for the University of California system for flying drones on campus property over 0.55 pounds. This experiment took place at the University of California, Santa Cruz (UCSC)

[1]Figure developed by Morgan Masters.

Westside Research Park (WRP) courtyard, a walled-in area with no risk of flying over people or damageable property, in full compliance with applicable laws and procedures.

4.1.1 Experiment: Odometry Enabled Polygon Flight

In this experiment, five AprilTags were placed in the designated flight zone at the WRP. Pins were placed on top of these tags via the handheld Clicker, saving each center as a RTK-enhanced GNNS point. A polygon based waypoint mission was flown over the targets where we recorded camera imagery, RTK enabled GPS points, IMU data, and odometry data. The flight lasted about 10 minutes, where two passthroughs occurred at a horizontal and vertical polygon orientation. Figure 4.2 illustrates a simplified visual map of what this mission looks like.

Once the data has been extracted from the Nano, it is passed through BirdsEye. The data is processed where each individual frame is isolated from the entire video and, using an AprilTag detector, we are able to detect the placed AprilTags. The AprilTag detector we used, *pupil-lab detector*[5], utilizes the the 36h11 AprilTag family which is harder to detect but minimizes the likelihood for false positives.

Utilizing BirdsEye spatial rotations, we are then able to associate the AprilTag centroids, acting as ground truthed points, to the pins. Comparing the difference in distance between the pins and the centroid we can infer the qualities of BirdsEye's accuracy. We expect to see an average difference of no more than 6cm between the centroid and the pins. At this level of accuracy BirdsEye can be deemed fit to be

[5]Github found here: `https://github.com/pupil-labs/apriltags`

utilized for precision agriculture applications.

Figure 4.4 is a visualisation of the imaging parsing step where we associate the pin and AprilTag locations in frame for each sampled step at 4Hz. This tools allows us to better assess the model's behavior as the vehicle changes dynamics and the frame changes shape. As the frame moves to see pins and AprilTags it drops a marker to signify where it thinks each object is.

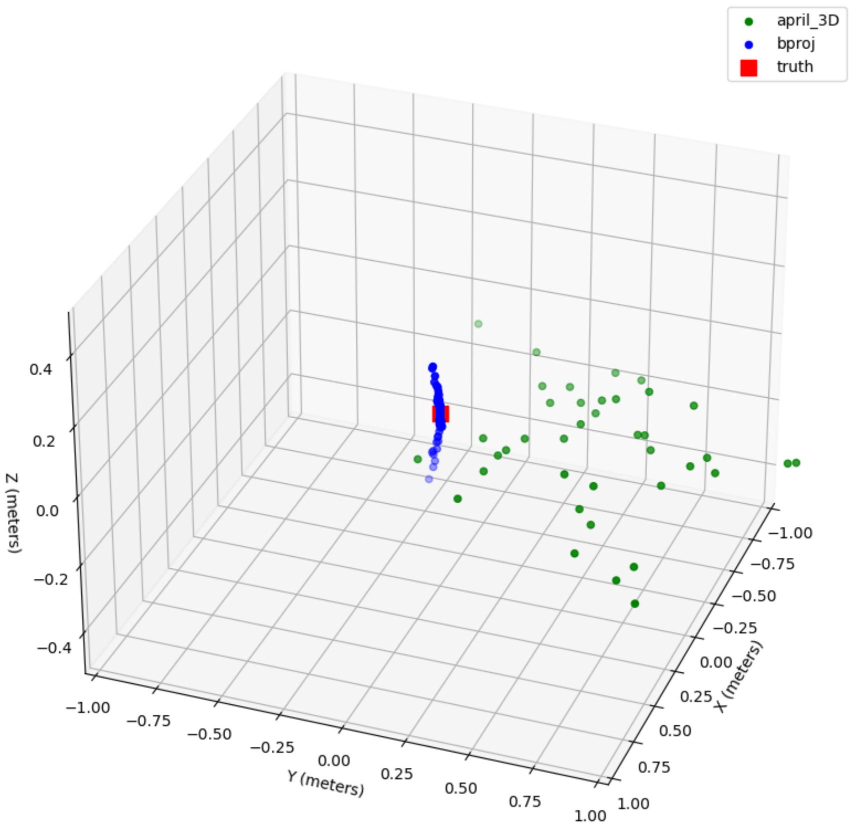


Figure 4.1: 3D map of where are pins are compared to their truths. While they should be overlapping there seems to be around a 1x1 meter difference between their positions. This indicates that there needs to be improvement with our spatial predictions.[1]

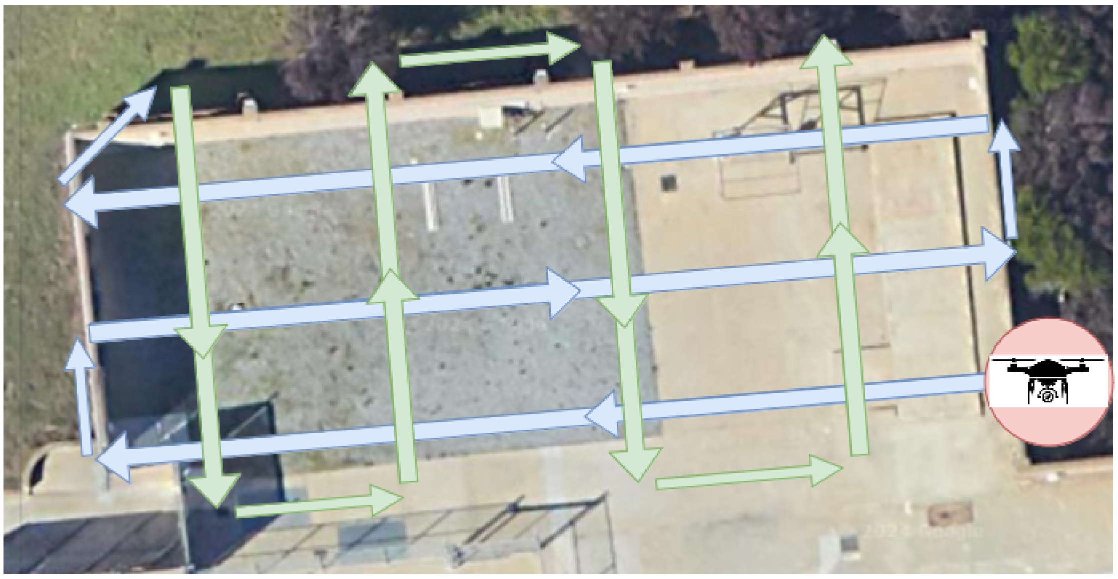


Figure 4.2: Polygon flights visualization. The drone begins at the red circle and flies the first horizontal polygon flight in blue. After finishing that orientation, it proceeds to fly the vertical polygon in green. AprilTags are randomly placed within this grid.

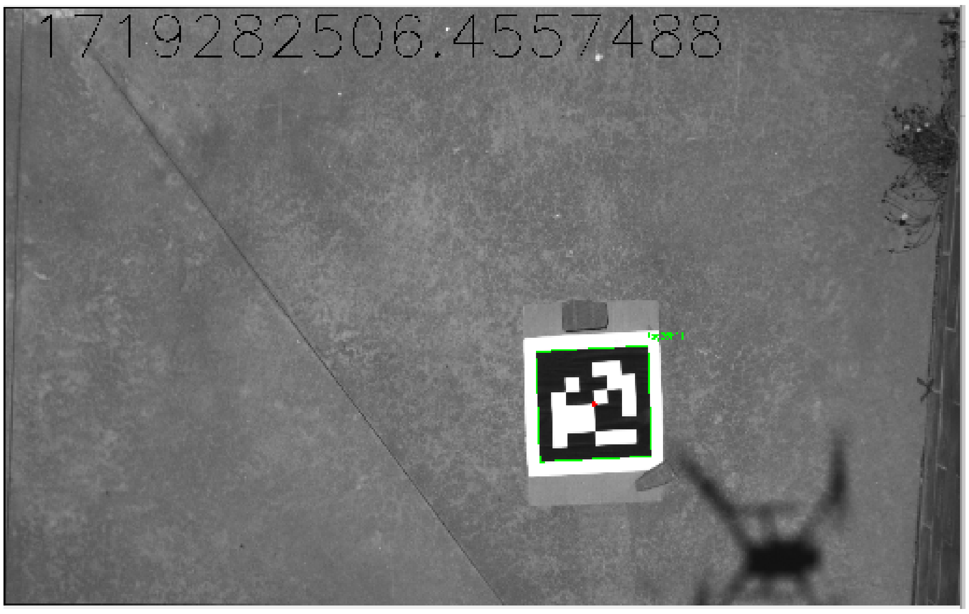


Figure 4.3: A sample image from the post processing pipeline where an AprilTag is detected. The image appears in the Mono8 grayscale format to help the AprilTag detector and user see the contour difference. The green outline around the Aprilag illustrates the detector's ability to recognize the tag and its spatial position in pixels. Using this information we can calculate the AprilTag centroid.

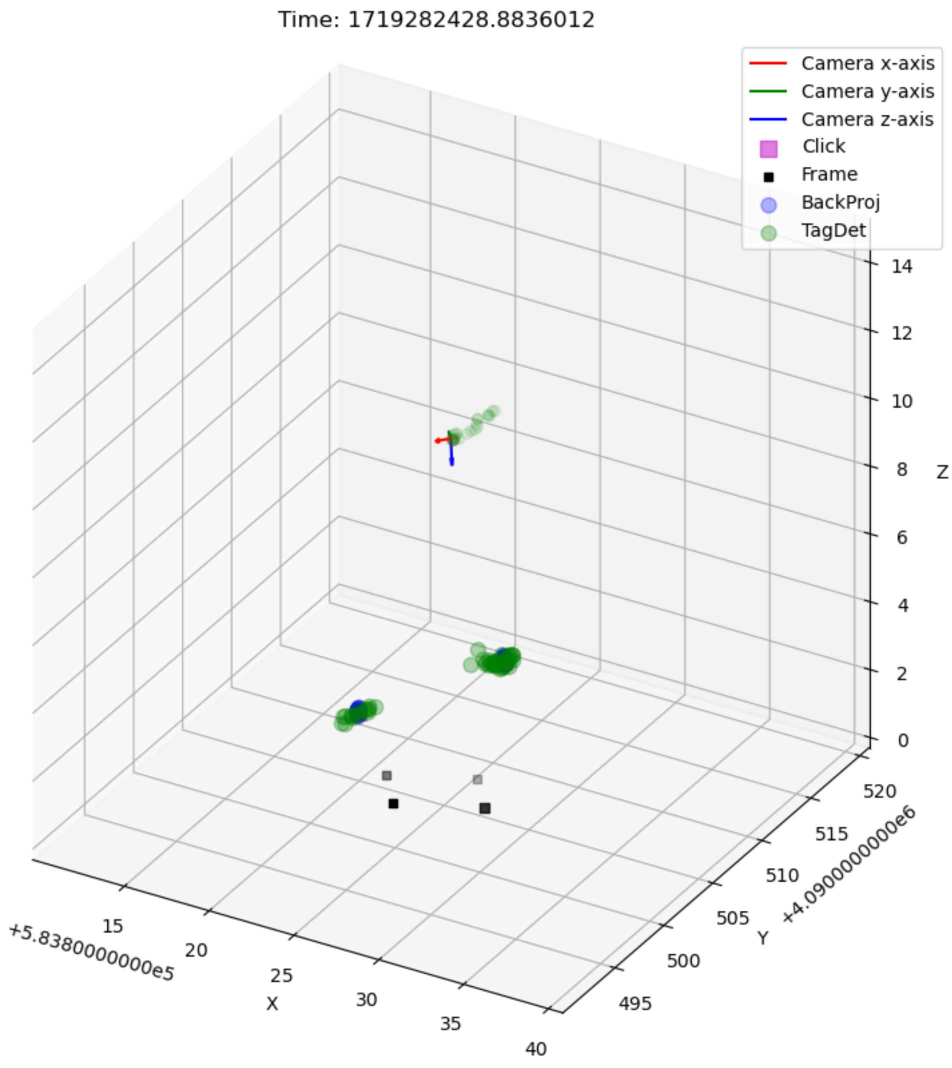


Figure 4.4: BirdsEye's post processing image parsing step. The triad flying at 10 meters is the drone with a trail of its previous positions labeled in green. A green trail signifies an RTK fix. Underneath the drone are gray and black squares which indicate the corners of the image frame. The pick squares are our ground truths pin locations which are eclipsed by its back-projections in blue and predicted AprilTag centroid in green.

4.2 Results

After conducting the experiment over the five AprilTags, we did not see the 6cm accuracy that we desired. BirdsEye's predicted imaging frame was not localized correctly with yet some spatial variances. This being said, swapping to the Blackfly, which has a better calibration and image fidelity, lead to better results compared to what we saw in Figure 4.1. This goes to show the negative impact that the previous MIPI cameras had on our system and the power of a well calibrated camera. This sections will outline our results, the spatial disparities that remain after implementing odometry and a better camera, and likely causes why these disparities appear.

4.2.1 Odometry Enabled Polygon Flight Results

Examining our results it is clear that odometry and a better camera and calibration was not the perfect solution to our issues. We still have a forward spatial shift in a majority of our detection that presumes that our positional estimation is still not quite correct. As seen with Figure 4.5 and Figure 4.1 there is a momentum-based offset that almost seems like we are sampling from the past. This incorrect alignment of time to space is likely due to our odometry node, which samples at 4Hz instead of a more moderate 10Hz. As we are capturing images at a faster pace and matching RTK to IMU samples at 10Hz, this is a likely explanation for some portion of the observed error.

Another considerable cause of error in this experiment was our RTK fix duration. Normally we obtain a 70% fix rate but in this experiment we only managed 40%.

This is due to the walled-in environment we flew in; we suspect ambient electromagnetic disturbances due to high voltage power lines that run underneath the flight area. The existence of these lines was only revealed late in the testing phase due to newly-initiated construction at the WRP facility. We suspect that our typical placement of base station in close proximity to these lines caused poor GPS reception and contributed to our errors. In total this resulted in an average increase of 15cm in our total AprilTag 3D projection accuracy compared to previous flights.

In figure 4.5, AprilTag positions are projected with our frame into 3D space and back to 2D where as the pins are just back-projected from 3D space to 2D space and associated in frame. As our heading travels along the x-axis, with a timing issue, our frame lags behind in the direction of our momentum leading our frame to be behind by sometimes more than a meter away. With our pins staying very close to the truth point we can say for certain that the initial 3D to 2D projection is what is suffering due to positional inaccuracies.

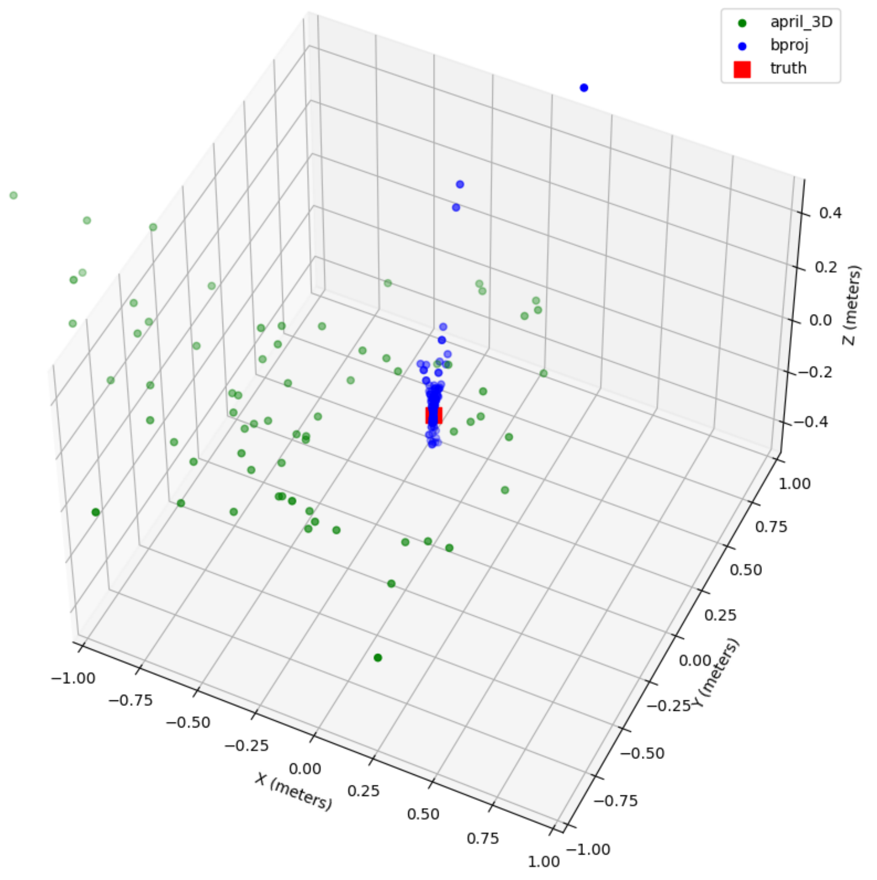


Figure 4.5: 3D back projected pin points to detected AprilTag locations. The green dots mark the detected AprilTag locations, the blue dots represent the back-projected pins, and the red square is the ground truth location. Ideally the green and blue dots would align, eclipsing each other on top of the red square.

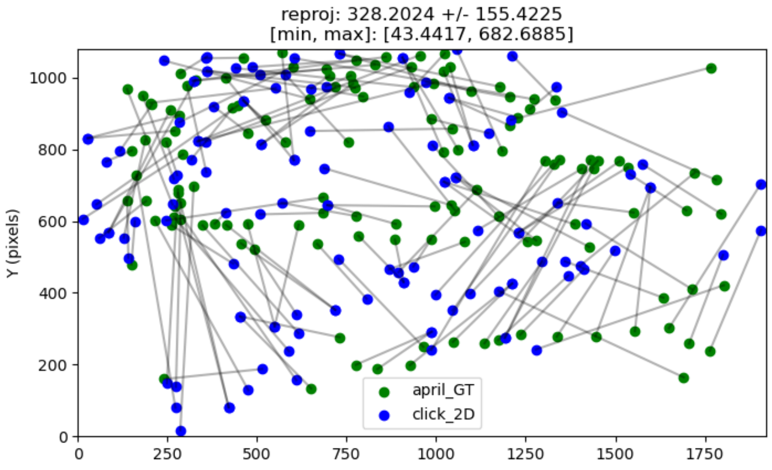
We can further see the behavior of our positional difference with Figure 4.6. These graphs illustrate the detail of the significant distances between both centroids and pins. In total we see an average pin reprojection error of 328.2 pixels or 8.6 centimeter between each pin to centroid frame pair. While this may seem close to our goal, the substantial amount of outliers can lead to wild inconsistencies with the BirdsEye system making them unsuitable for field usage.

Another valuable metric compares the undistorted to the raw images. While the figures above illustrate the proper undistorted data, Figures 4.7 and 4.8 show the data before our undistortion step. In total, we unsuprisingly due to our powerful camera, see little to difference in our reprojection error average being at 316.7 pixels or 8.37 centimeters, with a higher standard deviation. Where you can really see a change in the undistorted to raw features is between Figure 4.5 and 4.7 where the centroids are much further away from their actual geo-spatial locations. This is one of the advantages of image undistortion and rectification, where without it, our spatial understanding becomes warped. This goes to show that slight changes in camera parameters can make dramatics impacts on system performance.

Our final results are shown in Table 4.1. We were not able to achieve our desired results of a 6cm accuracy between our pins to ground truthed AprilTag centroids. Error in the odometry node induced by the RTK sampling rate impaired our results. In total, we see significant AprilTag 3D projection error in the x and y axes where the possibility to reach disparities to reach well over a meter is possible. Fortunately, our pin back-projection accuracy was much better with error reaching around 8.4cm.

Accuracy Results	
AprilTag 3D-projection accuracy (meters):	[-0.47259314 -0.22291539 0.0359816] +/- [0.53971824 0.4932817 0.07771641]
Click back-projection accuracy (meters):	[0.00291942 0.01191715 0.05065409] +/- [0.04094155 0.04978567 0.14320677]
Click Projection Error (pixels):	328.2024 +/- 155.4225

Table 4.1: BirdsEye projection accuracy results. Our AprilTag 3D reprojection accuracy struggles by having a average of 47cm accuracy in solely the x-direction with a standard deviation of 53cm. This being said our click reprojection accuracy was better with a 8.4cm error being close to our desired 6cm error.



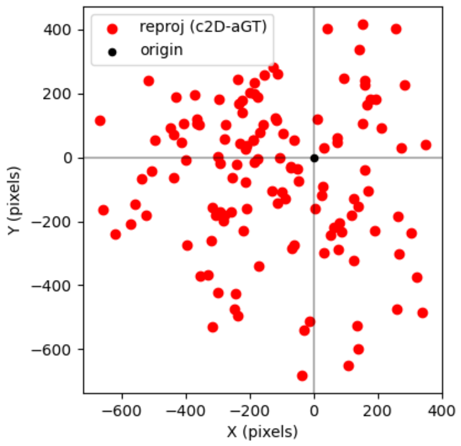


Figure 4.6: **Top:** AprilTag centroid and respective pin pairs with respect to each other. The blue dots represent the pins while the green dots represent the AprilTag centroids. Ideally, pairs would have on average a 6cm difference between each other. **Bottom:** all of our captured reprojected pins (in red) in frame if we centered the AprilTag centroid at the origin. Ideally our red dots would be significantly closer to or eclipsing the origin.

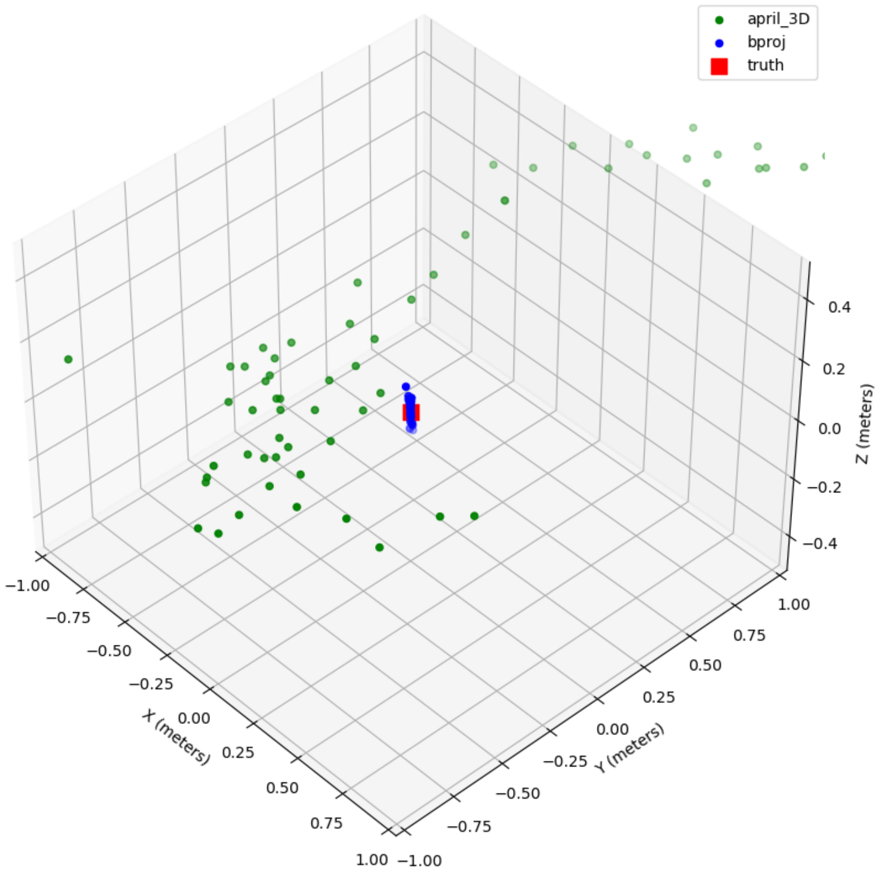
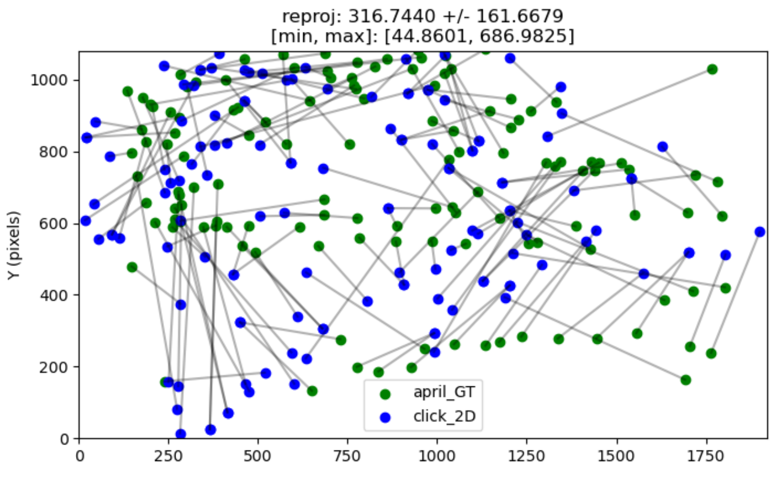


Figure 4.7: Distorted 3D back projected pin points to detected AprilTag locations. The AprilTag centroids shoot out much further compared to the undistorted counter part. This shows the importance of incorporating a undistortion step to correct image lens error.



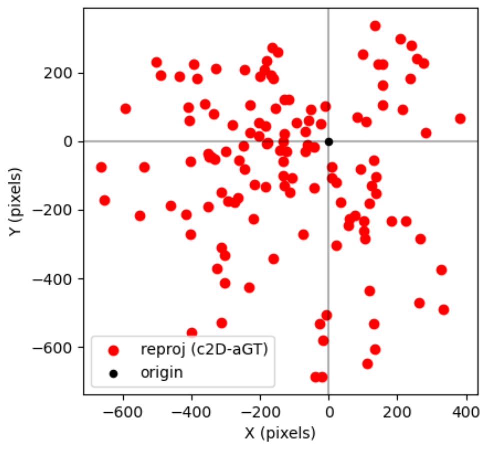


Figure 4.8: **Top:** distorted AprilTag centroid and respective pin pairs with respect to each other. **Bottom:** all of our distorted captured reprojected pins in frame if we centered the AprilTag centroid at the origin.

Chapter 5

Conclusion & Discussion

In the years to come, Precision Agriculture techniques will be paramount in order to maintain environmental sustainability, lower costs, and improve yield in farms. With the growing population, the demand for higher efficiency in food production is also increasing and gap the between farm and the technological state of the art must be narrowed. BirdsEye is a tool that has the potential to be widely used in diverse agricultural environments where the complexities of developing deep computer vision methods are simplified to be used for non-machine learning experts. This is done by creating a pipeline that allows for in-field labels to be used as annotations for visual datasets. Furthermore, this allows for the development of rapidly annotated datasets where the process to train autonomous systems to detect features such as plant stress, disease, gopher holes, dung, or any other visual feature becomes a lot more accessible. Unfortunately our results did not yield the accuracy that we desired. Our pins to ground truthed distances were well beyond the 6cm goal that we strived to achieve. At best, this work shows promise where a proof of concept of high accuracy is feasible with a little more time, effort, and planning to improve our estimation pipeline.

The work illustrated in this thesis was impaired by hardware and software-related challenges. In our seeming everlasting struggle to fix these issues, more hardware dysfunctions continued to present themselves until finally, during the making of this paper, we were permitted to replace our under performing hardware. The Jetson Nano was released in 2012; subsequent abandonment by NVidia with regards to Ubuntu and mainline Linux kernel support compromises the Nano's feasibility as a research platform. As a result of the evidence gathered through this thesis, several hardware upgrades are planned to upgrade CPU capability and improve timing characteristics. The GPS-IMU sensor fusion node will be further developed to enhance the accuracy of our attitude and position estimates. Unfortunately, during the planning stages of this project, we were too excited about developing a system that could be plausibly used by farmers instead of focusing on a well developed system that could be down-scaled once a proof of concept was achieved. What resulted from this vision is a system that was so heavily modified with custom features that it can not realistically be replicated even with its open sourced nature.

In the months to come, more accurate odometry and state estimation will be integrated into the system where we can obtain better positional corrections. Odometry is a critical component of improving BirdEye's accuracy and it's unfortunate that we realized this too late to properly implement by the making of this thesis. Despite our underwhelming resulting system performance, BirdsEye still shines as a concept that could improve and greatly bolster further research in the field of precision agriculture.

Chapter 6

Future Work

This work lays out the foundation for larger research and system wide improvements. BirdsEye can be used for any arbitrary vision based task that relies on aerial imagery, giving the power to non-machine learning experts to rapidly generate their own custom labeled datasets. The logical next step is to achieve our desired accuracy by implementing better state estimation in the form of more accurate odometry and undergoing hardware revisions.

The sensor payload system is planned for a revision going forward. Under preforming hardware is a critical concern in this project and we plan to redesign much of the system in order to support the longevity of this project's goals. The Jetson Nano is scheduled for a replacement to a Raspberry Pi 5 which has higher CPU capabilities and better community support. To finalize our MIPI camera replacements, a second Flir Blackfly camera without an IR filter will also be added. A camera without an IR filter is an important component to our system as it allows for the direct comparison between the visible spectrum and a visualized representation of the near-IR wavelengths that plants preferentially reflect. The 3D printed mount will also be designed to accommodate these hardware changes and promote better sensor accessibility and cable tidiness.

Once we have done these revisions and implemented more accurate odometry, we will have achieved significantly better positional corrections and BirdsEye accuracy. We then plan to train a deep CNN utilizing BirdsEye generated annotated datasets. As we explore these preliminary stages we decided upon building out the DeepLabv3 architecture using a semi supervised concept. This implementation will allow us to build practical datasets where perhaps not all desired field features are labeled. To validate this training pipeline we plan to conduct a few more validations tests on stationary field targets. Gopher hole detection and plant speciation are among these tests where we can examine our accuracy and classification ability on small compact targets that never change position.

Going further with this work, we ultimately plan to develop large scale datasets that conduct disease and stress based identification in local agricultural environments such as the UC Santa Cruz Center for Agroecology (CfA), a 30 acre organic farm with the mission to advance agroecology and equitable food systems by providing a venue of research and collaboration. This will be a collaborative endeavor where clickers would be handed to farm staff who have the expertise to classify plant diseases and stress based symptoms on the ground. Drone flights could subsequently be preformed and stages of diseases could be tracked throughout the changing seasons. Over time we could obtain enough data to train a deep learning pipeline to autonomously identify desired symptoms. Going further, once we can recognize areas with a higher concentration of stress, we could then optimize our flight route with a path planning system that calculates the most battery efficient route to image regions of interest. This way our 30 minute flight time can be more efficiently used in large acre farms.

There is also plans for revising some of BirdsEye's functions. The flat-Earth assumption is an inaccurate estimate that muddles our projection capabilities. Therefore, we plan to utilize Digital Elevation Maps (DEMs) to have a better understanding of the terrain we are flying over. With these DEMs our estimates of the terrain will allows create more spatially realistic rotations between the 2D and 3D world. Some DEMs already exist of our local area, however, we eventually plan to develop our maps using lidar based imaging systems that will allow our elevation estimation to go from meter accuracy to centimeter accuracy. This will allow us to fly in more complex areas while retaining minimal amounts of error.

Once our system has been thoroughly developed we plan to coordinate joint robot operations with a Unitree B1 quadruped robot equipped with visual and lidar based sensing capabilities. With ground and aerial based sensing modalities, we would be able to achieve high precision detections and visualization of our desired environment. One such project is generating high fidelity 3D scene reconstructions from joint robot imagery. With this information we could extract previously recorded imagery and play back a full 3D reconstruction of the given environment. Having multi-perspective imaging data will cover robot blind spots and allow to us to achieve high quality reconstructions, usable for both online planning and offline visualizations.

Bibliography

[1] Remote sensing and geo-information technologies in agriculture. https://seos-project.eu/agriculture/agriculture-c01-s02.html. (Accessed on 02/25/2024).

[2] Chapter 1. Digital image representation. http://pippin.gimp.org/image_processing/chap_dir.html, September 2017. [Online; accessed 27. May 2024].

[3] Vehicle State Estimation on a Roadway. https://jasleon.github.io/Vehicle-State-Estimation, June 2021. [Online; accessed 7. Jun. 2024].

[4] Loss Functions — ML Glossary documentation, July 2022. [Online; accessed 31. May 2024].

[5] Accelerate Convergence: Mini-batch Momentum in Deep Learning. https://statusneo.com/accelerate-convergence-mini-batch-momentum-in-deep-learning, September 2023. [Online; accessed 1. Jun. 2024].

[6] AllanVarianceExample. https://www.mathworks.com/help/fusion/ug/inertial-sensor-noise-analysis-using-allan-variance.html, June 2024. [Online; accessed 6. Jun. 2024].

[7] CS 230 - Convolutional Neural Networks Cheatsheet. https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-convolutional-neural-networks, May 2024. [Online; accessed 28. May 2024].

[8] Differences Between Epoch, Batch, and Mini-batch | Baeldung on Computer Science. https://www.baeldung.com/cs/epoch-vs-batch-vs-mini-batch, March 2024. [Online; accessed 31. May 2024].

[9] How RTK works | Reach RS3. https://docs.emlid.com/reachrs3/rtk-quickstart/rtk-introduction, May 2024. [Online; accessed 16. May 2024].

[10] Jetson Nano. https://developer.nvidia.com/embedded/jetson-nano, June 2024. [Online; accessed 15. Jun. 2024].

[11] Model Fit: Underfitting vs. Overfitting - Amazon Machine Learning. https://docs.aws.amazon.com/machine-learning/latest/dg/model-fit-underfitting-vs-overfitting.html, May 2024. [Online; accessed 30. May 2024].

[12] REP 105 – Coordinate Frames for Mobile Platforms (ROS.org). https://www.ros.org/reps/rep-0105.html, June 2024. [Online; accessed 17. Jun. 2024].

[13] Supervised, unsupervised, and semi-supervised ML | Glossary. `https://www.usertesting.com/glossary/s/supervised-unsupervised-and-semi-supervised-machine-learning-ml`, May 2024. [Online; accessed 31. May 2024].

[14] What is a Neural Network? | IBM. `https://www.ibm.com/topics/neural-networks`, May 2024. [Online; accessed 27. May 2024].

[15] Abien Fred Agarap. Deep Learning using Rectified Linear Units (ReLU). *arXiv*, March 2018.

[16] Skann. ai. Projecting 3D Points into a 2D Screen - Skann.ai - Medium. *Medium*, September 2023.

[17] Cabe Atwell. ROS 2 Explained: Overview and Features. *Electronic Design*, January 2022.

[18] Andrea Botta, Paride Cavallone, Lorenzo Baglieri, Giovanni Colucci, Luigi Tagliavini, and Giuseppe Quaglia. A review of robots, perception, and tasks in precision agriculture. *Applied Mechanics*, 3(3):830–854, 2022.

[19] Jason Brownlee. Understand the Impact of Learning Rate on Neural Network Performance - MachineLearningMastery.com. *MachineLearningMastery*, September 2020.

[20] Kevin Crowston. Amazon Mechanical Turk: A Research Tool for Organizations

and Information Systems Scholars. In *Shaping the Future of ICT Research. Methods and Approaches*, pages 210–221. Springer, Berlin, Germany, 2012.

[21] Drone Launch Academy. How Are Drones Used in Agriculture? *Drone Launch Academy*, May 2024.

[22] Jorge Gago, Cyril Douthe, Rafael E Coopman, Pedro Pablo Gallego, Miquel Ribas-Carbo, Jaume Flexas, Jb Escalona, and Hb Medrano. Uavs challenge to assess water stress for sustainable agriculture. *Agricultural water management*, 153.

[23] Francisco Garcia-Ruiz, Sindhuja Sankaran, Joe Mari Maja, Won Suk Lee, Jesper Rasmussen, and Reza Ehsani. Comparison of two aerial imaging platforms for identification of huanglongbing-infected citrus trees. *Computers and Electronics in Agriculture*, 91:106–115, 2013.

[24] Suraj G Gupta, Dr Mangesh Ghonge, and Pradip M Jawandhiya. Review of unmanned aircraft system (uas). *International Journal of Advanced Research in Computer Engineering & Technology (IJARCET) Volume*, 2, 2013.

[25] Samuel C Hassler and Fulya Baysal-Gurel. Unmanned aircraft system (uas) technology and applications in agriculture. *Agronomy*, 9(10):618, 2019.

[26] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*, pages 630–645. Springer, 2016.

[27] Melissa G. Keith, Louis Tay, and Peter D. Harms. Systems Perspective of Amazon Mechanical Turk for Organizational Research: Review and Recommendations. *Front. Psychol.*, 8:274407, August 2017.

[28] Gopal Krishna, Rabi N. Sahoo, Prafull Singh, Vaishangi Bajpai, Himesh Patra, Sudhir Kumar, Raju Dandapani, Vinod K. Gupta, C. Viswanathan, Tauqueer Ahmad, and Prachi M. Sahoo. Comparison of various modelling approaches for water deficit stress monitoring in rice crop through hyperspectral remote sensing. *Agricultural Water Management*, 213:231–244, March 2019.

[29] Michael. Semantic Segmentation vs Object Detection: Differences. *Keymakr's Blog features the latest news and updates*, March 2024.

[30] Sharada P Mohanty, David P Hughes, and Marcel Salathé. Using deep learning for image-based plant disease detection. *Frontiers in plant science*, 7:215232, 2016.

[31] Nirmala Murali. Image Classification vs Semantic Segmentation vs Instance Segmentation. *Medium*, January 2022.

[32] Aaron Nathan. How to Build Your Own RTK Base Station (& Is It Worth It?) [2024] | Point One Navigation. *Point One Navigation*, February 2024.

[33] Krishna Neupane and Fulya Baysal-Gurel. Automatic Identification and Monitoring of Plant Diseases Using Unmanned Aerial Vehicles: A Review. *Remote Sensing*, 13(19):3841, January 2021. Number: 19 Publisher: Multidisciplinary Digital Publishing Institute.

[34] M. Ghelich Oghli. Digital image processing in the name of god digital image processing lecture6: Color image processing. `https://slideplayer.com/slide/8752313`, May 2024. [Online; accessed 27. May 2024].

[35] Lucas Prado Osco, Ana Paula Marques Ramos, Érika Akemi Saito Moriya, Lorrayne Guimarães Bavaresco, Bruna Coelho de Lima, Nayara Estrabis, Danilo Roberto Pereira, José Eduardo Creste, José Marcato Júnior, Wesley Nunes Gonçalves, Nilton Nobuhiro Imai, Jonathan Li, Veraldo Liesenberg, and Fábio Fernando de Araújo. Modeling Hyperspectral Response of Water-Stress Induced Lettuce Plants Using Artificial Neural Networks. *Remote Sensing*, 11(23):2797, January 2019. Number: 23 Publisher: Multidisciplinary Digital Publishing Institute.

[36] Richard R Picard and R Dennis Cook. Cross-validation of regression models. *Journal of the American Statistical Association*, 79(387):575–583, 1984.

[37] Francis J Pierce and Peter Nowak. Aspects of precision agriculture. *Advances in agronomy*, 67:1–85, 1999.

[38] Sabina Pokhrel. Image Data Labelling and Annotation — Everything you need to know. *Medium*, December 2021.

[39] Lia Reich. Understanding your Aerial Data: Normalized Difference Vegetation Index NDVI - Geoawesomeness. `https://geoawesomeness.com/eo-hub/`

understanding-aerial-data-normalized-difference-vegetation-index-ndvi, April 2024. [Online; accessed 2. Jun. 2024].

[40] Sumit Saha. A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way. *Medium*, April 2023.

[41] Xiaodong Song, Ganlin Zhang, Feng Liu, Decheng Li, Yuguo Zhao, and Jinling Yang. Modeling spatio-temporal distribution of soil moisture by deep learning-based cellular automata model. *Journal of Arid Land*, 8:734–748, 2016.

[42] Farmonaut Support. Applications of satellite imagery bands part 2: Vegetation red edge (b5,b6,b7,b8a). https://medium.com/@farmonaut/applications-of-satellite-imagery-bands-part-2-vegetataion-red-edge, Jan 2019. (Accessed on 02/22/2024).

[43] Gerard Sylvester. E-agriculture in action: drones for agriculture. 2018.

[44] Yahya Tawil. An Introduction to Robot Operating System (ROS&#41. *All About Circuits*, March 2020.

[45] Edna Chebet Too, Li Yujian, Sam Njuki, and Liu Yingchun. A comparative study of fine-tuning deep learning models for plant disease identification. *Computers and Electronics in Agriculture*, 161:272–279, 2019.

[46] Luis Von Ahn, Benjamin Maurer, Colin McMillen, David Abraham, and Manuel Blum. recaptcha: Human-based character recognition via web security measures. *Science*, 321(5895):1465–1468, 2008.

[47] Rui Xu, Changying Li, and Andrew H. Paterson. Multispectral imaging and unmanned aerial systems for cotton plant phenotyping. *PLoS One*, 14(2):e0205083, February 2019.

[48] Yunseop Kim, David M Glenn, Johnny Park, Henry K Ngugi, and Brian L Lehman. Hyperspectral Image Analysis for Plant Stress Detection. In *2010 Pittsburgh, Pennsylvania, June 20 - June 23, 2010*. American Society of Agricultural and Biological Engineers, 2010.

[49] Nanyang Zhu, Xu Liu, Ziqian Liu, Kai Hu, Yingkuan Wang, Jinglu Tan, Min Huang, Qibing Zhu, Xunsheng Ji, Yongnian Jiang, et al. Deep learning for smart agriculture: Concepts, tools, applications, and opportunities. *International Journal of Agricultural and Biological Engineering*, 11(4):32–44, 2018.