The Farm Owl

An agricultural survey in autonomous flight and computer vision

Mason Reyher

Cole Orelup

Introduction

An agricultural combine contains more computers and technology inside it than modern rocket ships. Why? Because civilization does not depend on space travel. Civilization does, however, depend on food. Despite the extraordinary amount of convenience and efficiency modern computing brings to the table for general purpose, it will always be more useful to agriculture.

Our goal is to create a drone that will reduce fertilizer and pesticide use in modern farming. This will be accomplished by flying over farmland autonomously, taking pictures every so often using an onboard camera. The microcomputer on the drone will then compile these pictures into one large image. In the future, the final image will be sent through a machine learning model for inference. The inference results will show where a farm specifically needs fertilizer/pesticides. Since this project encompasses so many disciplines (computer vision, data engineering, remote sensing, flight, machine learning), we will focus our efforts on creating the autonomous drone and image stitching algorithm as our main priority/ minimum viable product (MVP).

Remote sensing drones have been created before. They scan farmland from the skies by utilizing different wavelengths of the electromagnetic spectrum.[1] Weeds and pests give off different spectrums of light when examined - allowing them to be identified and destroyed via precision agriculture. The problem with this system is that it is an expensive one.[2] These advanced agriculture drones can cost upwards of ten-thousand dollars.

The drone we are creating will have three goals. The first is to be inexpensive. Technology is useless if it only goes to the highest bidder. A cheap drone (around $100-$250) would be useful to a farmer and would not be a burdensome investment like a tractor would be.

[1] John Nowastski, Agricultural Remote Sensing Basics
[2] Censys Technology Drones

The second goal is for the drone to be under 0.55 pounds, which is the weight threshold for personal drones. An autonomous flying vehicle that is under 0.55 pounds does not have to be registered, making it easier to use for a farmer, who has more important tasks to focus on.[3]

The final goal of this project is to create a free, unlicensed image stitching algorithm. The image stitcher is used to turn a previous streaming process into a batch process. Since the drone needs to be lightweight, there can not be a lot of memory attached to the computer. A streamed video as the drone flies over would take up far too much precious memory. However, several images taken over a few minutes would be manageable. The image stitching algorithm will take these pictures and mesh them together to create one, high DPI image, that can be fed to a machine learning model for inference. These types of algorithms exist, but they are expensive and locked up inside academia. Ours will be free for use by precision agriculturists - saving money for them and lowering the cost of commercial ag drones.

Overall, the drone we create will be lightweight, cheap, and open-source. Our goal is not to sell this product, but show that this product is possible. This kind of technology could drastically reduce the costs for farmers and beyond, all the way to consumers.

[3] Federal Drone Registration

Cole Orelup

Contact

coleorelup@gmail.com
(406) 702-6017

Education

Major: Computer Science
Minor: Data Science
Overall GPA: 3.65
08/2019 - Present
Expected Graduation: May 2023

Skills

Languages: Python, JavaScript, Java, C
Tools: VSCode, Jetbrains, Git, Github
Abilities: Flexibility, Communication, Leadership

Awards

- Dean’s List Spring 2021
- Dean’s List Spring 2022
- President’s List Fall 2021

Experience

React Front-End Developer Summer 2022
Colorado Avalanche Information Center (CAIC)
Bozeman, MT

- Used React to create dynamic components for CAIC’s new avalanche awareness website
- Followed design mockups created by CAIC
- Worked with a Drupal backend

Machine Learning Fall 2022
Montana State University
Bozeman MT

- Worked on implementing various machine learning methods from scratch
- Gained experience with multilayer perceptrons and its related algorithms
- Data Handling

Mason Reyher

303-918-6558 • mr.noitall@yahoo.com • linkedin.com/in/mason-reyher • github.com/koalafant

EDUCATION

Montana State University — **Bozeman, MT**
B.S. Computer Science, Environmental Horticulture — **May 2023**
GPA: 3.86

CERTIFICATIONS

● **Google Cloud Platform Professional Data Engineer**	**August 2022**
● **AWS Machine Learning - Specialty**	**July 2022**
● **AWS Data Analytics - Specialty**	**June 2022**
● **AWS Security - Specialty**	**June 2022**
● **AWS Solutions Architect - Associate**	**May 2022**
● **AWS Cloud Practitioner**	**April 2022**

SKILLS

- **Languages: Python, SQL Java, C, C++**
- **Technology: Linux, Bash, Git, Agile, Docker**
- **Cloud: GCP, Azure, AWS**
- **Data engineering: dbt, ELT, MYSQL, Terraform, NoSQL, BigQuery**
- **Apache: Airflow, Beam, Spark, Kafka, Hadoop**

WORK EXPERIENCE

Precocity LLC — **Bozeman, MT**
Data Engineer — **Aug 2022 – Present**
- **Building out dbt models with BigQuery based SQL and Python Jinja**
- **Refactoring and developing Terraform scripts for Google Cloud infrastructure**
- **Creating and maintaining ELT pipelines**
- **Writing DAGs in Composer hosted Apache Airflow**
- **Maintaining a CI/CD pipeline using Azure Devops**

Montana State University — **Bozeman, MT**
Undergraduate Teaching Assistant — **Jan 2022 – May 2022, Aug 2022 - December 2022**
- **Explained object-oriented programming and functional programming**
- **Taught Python and Java to undergraduate students**

Amazon Web Services (AWS) — **Herndon, VA**
Cloud Consultant Intern — **May 2022 - Aug 2022**
- **Created a data and inference pipeline to offer the training, validation, and inference of an arbitrary number of object detection machine learning models in a single click, using Amazon Sagemaker, Lambda, API Gateway, S3, and Docker**
- **Utilized YOLO, FasterRCNN, and RCNN image classification algorithms**
- **Designed a semantic segmentation algorithm from scratch for exploratory data analysis in the COCO dataset**

Techlink — **Bozeman, MT**
System Administrator Intern — **Sep 2021 – May 2022**
- **Configured MDM and endpoint solutions in Azure**
- **Promoted digital security through group policy and Azure Active Directory**
- **Created effective IAM solutions in a hybrid cloud environment**

Background

Algal Blooms:

An algal bloom results from the exponential growth of cyanobacteria, which thrives in the excess nutrients (mainly nitrogen and phosphorus) present in water. This results in the cyanobacteria (algae) consuming itself to death, leading to high levels of cyanotoxins in the water called hypoxia[4]. By themselves, these toxins are deadly and eat up all the oxygen present in water. Meaning, sealife -both plants and animals- can not survive. This is why algal blooms are also called dead zones - there is no living being in the water. The Gulf of Mexico dead zone is 6334 square miles as of 2021[5], and is a byproduct of farmers using too much fertilizer in the Golden Triangle. This fertilizer has gradually been washed down the Mississippi river into the Gulf of Mexico.

Raspberry Pi Zero:

The Raspberry Pi Zero is one of the smallest single board computers (65mm x 30mm, 5mm) available for general purchase. It comes with 512MB of ram, a CSI camera connector, and a 1Ghz single-core CPU.[6] This board should be powerful enough to control the flight and camera systems on the drone. Additionally, the board is lightweight (0.56 oz), making it suitable for flight use.

CAD (Fusion 360):

Computer Aided Design, or CAD, is used to sketch, model, and create objects and tools for use in mechanical engineering and related disciplines. We used CAD to develop a drone frame to mount and center all the hardware, as well as provide a light frame to aerodynamically produce lift with minimal effort.

Python OpenCV:

OpenCV (Open Source Computer Vision) is the biggest CV module that anyone can use. It supports multiple computer languages (Python in our case), and thousands of algorithms for use in image analysis.[7] Since

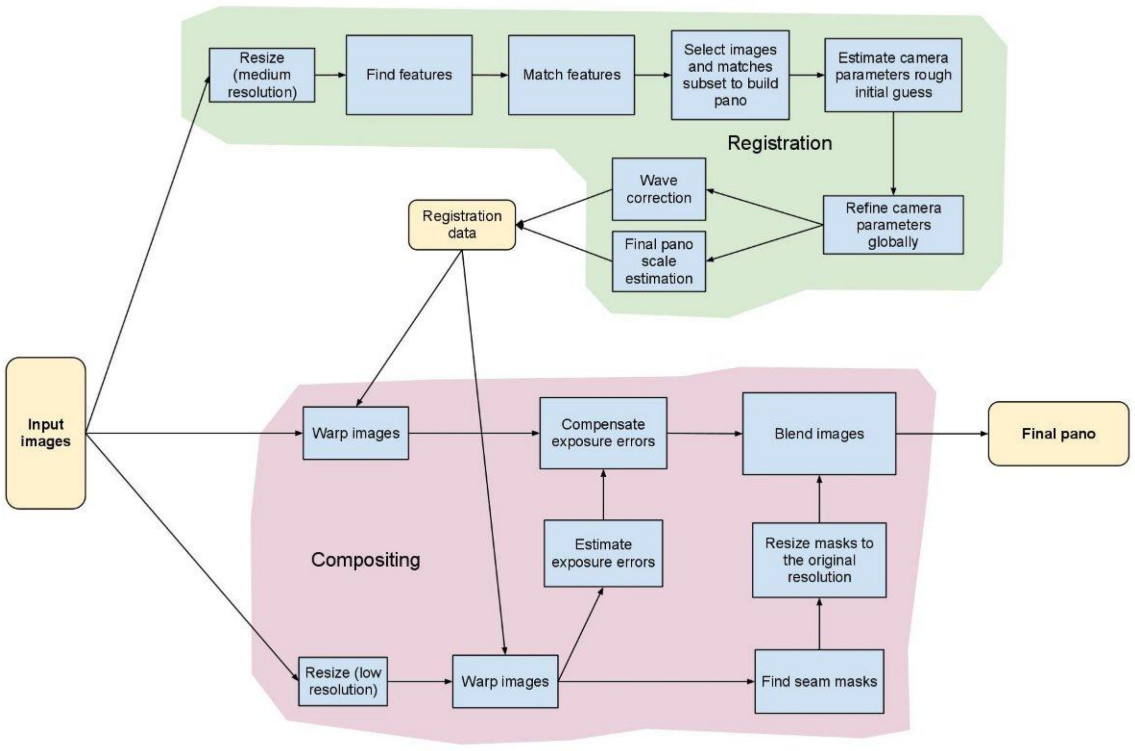
[4] National Geographic: Dead Zones
[5] NOAA: 2021 Gulf of Mexico Dead Zone
[6] Raspberry Pi Zero
[7] OpenCV: Python Basics

Raspberry Pi supports Linux operating systems, and by transition, Python, we can utilize OpenCV-Python to manipulate the images our drone captures.

Image Stitching:

Overview:

Image stitching is a computer vision algorithm that is used to combine multiple digital images together, with a goal of increasing overall definition of field of view (FOV)[8]. Our autonomous drone can use image stitching techniques to reduce the uptime of our onboard camera. By taking snapshots every few seconds (calculated by average velocity and size of flight path), the images can be compiled after flight time and stitched together to offer one large-scale image of the farm instead of a continuous video. This image will serve as a datapoint for future machine-assisted analysis.


Resize (medium resolution)
Find features
Match features
Select images and matches subset to build pano
Estimate camera parameters rough initial guess
Registration
Wave correction
Registration data
Refine camera parameters globally
Final pano scale estimation
Input images
Warp images
Compensate exposure errors
Blend images
Final pano
Compositing
Estimate exposure errors
Resize masks to the original resolution
Resize (low resolution)
Warp images
Find seam masks

[8] S Mann, R Picard: Virtual Bellows

Feature Matching:
The way people used to assess fingerprints in crime scenes was using something called minutia. Minutiae are like the parts that don’t match up. In a traditional loop fingerprint, someone with a scar or bump in the middle of their loop would have a minutia there. Fingerprint sensors in cell phones find these minutia and create polygons out of them. If a great many polygons match up to the stored record of your fingerprint, then you are allowed access. Feature Matching works exactly the same way, but with an image. It finds the minutia of an image and returns its location. If you find the same minutia in two images (these minutia are called features), then you can likely stitch the image together at that point.

Camera Estimation:
Due to the nature of a panorama, an image must coalesce at one spot. Someone taking the photo cannot be in two places at once (for the most general of examples). If, however, a photographer does move while taking the panorama (or a drone takes pictures from different points in the sky) the camera position has moved. Camera estimation attempts to find a common ground between moments where pictures have been taken, and using warping algorithms (provided by OpenCV), the image can be warped around that central moment, creating a continuous, or panoramic effect. Camera estimation is crucial to image stitching since two moments stitched together without being coalesced will produce lopsided images (much like different projections on maps).

Blending:
Once accurate seam masks (derivatives of the features above) have been calculated and images have been warped around consistent moments (camera estimation above), the final image can be blended and ‘stitched’ together. This process involves small amounts of linear algebra and blurring techniques to create a continuous feeling. Optional exposure correction can also occur using techniques like histogram equalization, but those techniques are out of scope for this project.

Potential Algorithms:

Unsupervised Learning Algorithms:

Unsupervised learning is a type of machine learning (subset of artificial intelligence) that heavily relies on patterns. These algorithms are given large amounts of unclassified data and can be used in classification and regression scenarios. Oftentimes, there are manually tunable variables called hyperparameters that assist learning the underlying patterns in data.[9]

Random Cut Forest:

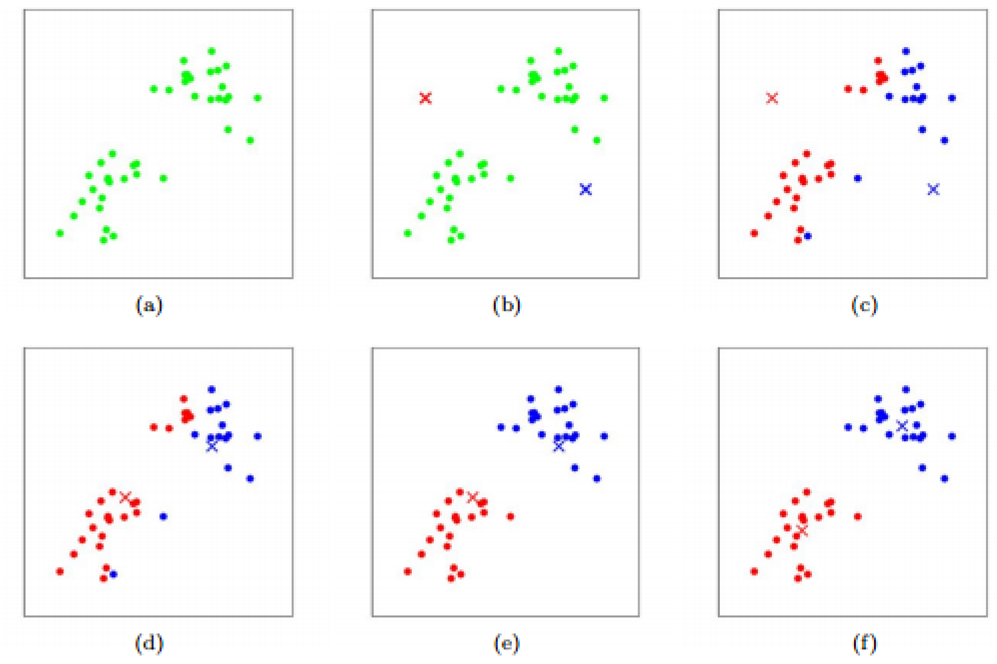
Random Cut Forest (RCF) is an unsupervised algorithm that can detect outlier/anomalous data within datasets.[10] It does this by essentially "cutting" the data like a pizza, in random spots. The sparsest section in the resulting dataset is dubbed with an anomalous score. This can be used in this project with RGB data from images of farms. Pixels that are more yellow or grayish than others would receive a higher (worse) score than the greener and darker plants.

K-Means:

K-Means is an algorithm primarily for identifying clusters in a dataset. In this instance, "K" corresponds to the number of clusters that the algorithm attempts to find and the "Means" corresponds to the centroids, or centers, of the clusters. K-Means first starts by picking K random points and initializes them as a centroid. It then computes the distance (usually euclidean distance) between points in the data and the centroids. These new data points become a part of the closest centroid's cluster. Afterwards, the mean distance of each cluster is then calculated and the respective centroid is reassigned to the mean value. This process is repeated for a defined length or until the clusters do not change. The set of clusters with the optimal total variance is returned.

The image below[11] shows the process of K-Means. The X's are the centroids and the dots are the data points.

[9] IBM: Unsupervised Learning
[10] Sijin Yiom, Jae-Hung Jung: Random Cut Forest
[11] Chris Piech: K-Means



This type of algorithm is useful in unsupervised learning scenarios where the labels of the dataset are unknown.

Supervised Learning Algorithms:

Supervised learning is a type of machine learning that involves labeled data. The algorithms are given input data and are expected to make predictions about data. Unlike unsupervised learning, supervised learning algorithms[12] are trained on data that they are "given the answer to." They then make predictions based on their training, and if they are wrong, they adjust weights to get closer to the "correct answer."

Perceptron:

A perceptron is a single layer of artificial neurons[13]. Picture a wall of lights: the perceptron will light up when something touches a sensor on the other side[14]. Using this, if you were to put your hand on the sensors, the perceptron would light up in a hand-like shape.

[12] IBM: Supervised Learning
[13] Mayank Banoula: Perceptrons
[14] Scott Pletcher: Intro to Machine Learning

Convolutional Neural Network:

A convolutional neural network (CNN) is a type of multi-layer perceptron that is versatile and can be used for image classification tasks. The supervised nature of this algorithm requires numerous annotated data be fed to train. A number of weights and nodes reside inside the network, and are activated (much like a neuron in the brain) based on the presence of some stimulus in the input (an image in our case).[15] The input will proceed through a predefined number of layers, until an eventual answer is given (classification, regression number, etc.). The network will then train itself using a loss function (either Stochastic Gradient Descent (SGD) in our case, or more modern algorithms like Adam)[16] and backpropagation. A CNN can be used to input the stitched farm image, and regression can yield scores for sections of the farmland. A RCNN (below) can be much more useful for this type of analysis.

Region-Based Convolutional Neural Network:

A region-based convolutional neural network (RCNN) is a type of scanning neural net. The key difference between a CNN and RCNN is that an RCNN can not only classify an object, but it can classify *where* the object is.[17] Using this type of network (or an idea similar to it), we can break a farm plot into various regions to identify which crop areas are struggling the most.

Ensemble Learning:

Ensemble learning is a modern approach to machine learning. It is the process of combining multiple predictions from multiple different ML models. The application we can use is combining the RCF model and an RCNN, to determine anomalous areas of farmland where the crops are visually struggling. Ensemble learning (in theory) can add great accuracy and precision to predictions,[18] and in our case can act like a filter (different shades of green should not be weighted as much as a dead crop patch).

---

[15] IBM: Convolutional Neural Networks
[16] Aman Gupta, et al. SGD vs. ADAM
[17] Rohith Gandhi: R-CNN, Fast R-CNN, Faster R-CNN, YOLO
[18] Jason Brownlee: Introduction to Ensemble Learning

Work Schedule

Milestone Schedule:

Due Date	Milestones
January 30	Purchase hardware
February 6	Airframe built
February 13	1st Flight: Hover 1m for 10s
February 20	Camera installed and functioning
February 27	Camera Driver complete
March 6	Image capture algorithm complete
March 13	Drone flight driver test: 1m forward, 1m backward, 1m left, 1m right, land
March 13	Image stitching algorithm complete
March 27	Image stitcher and image capture integration
March 27	Drone driver integrations
April 25	Print Poster
April 27	MSU Student Research Celebration
May 1	Proposal Due
May 2	Class Presentation

Software Development Life Cycle (SDLC):
Farm Owl will be using a fairly standard agile and scrum methodology. Meaning, the project will be broken into two week long sprints where development is done incrementally in small, manageable chunks. A meeting will take place at the beginning and end of each sprint. Sprint planning occurs at the beginning and plans out what needs to

happen over the span of the sprint. The sprint review and retrospective at the end, includes a demonstration of the work accomplished and what could be improved upon in the next sprint. In addition to these more involved meetings, short standups will occur every Tuesday and Thursday of the week. Traditionally, a standup occurs every day, however for the purposes of this project, twice a week should suffice. This allows for adequate work to be done, which should allow for higher quality standups. Agile and scrum is an ideal SDLC for this project because designing a drone is an iterative process. Trying to throw something together all at once without regard for the smaller components is a setup for failure. The project needs to be done gradually so that it is 1) achievable and 2) a quality product. Agile and scrum both promote this type of behavior in project management, which is exactly why it is being used as the SDLC for Farm Owl.

Sprint Schedule:

Time Frame	**Goals**
Sprint 1: January 30 - February 13	● Hover-worthy airframe put together ● Wiring done ● Basic elevation controls
Sprint 2: February 13 - February 27	● Forward, backward, left, right movement ● Camera installed on airframe ● Camera driver functioning
Sprint 3: February 27 - March 13	● Start and complete image capturing ● Start and complete image stitching ● Test flight driver
Sprint 4: March 13 - March 27	● Integrate image capturing and image stitching ● Test camera capturing and stitching
Sprint 5: March 27 - April 10	● Integrate drone driver ● Drone-wide bug fixes and optimizations
Sprint 6: April 10 - April 24	● Overall drone performance testing with flight and camera ● Finishing touches
Sprint 7: April 24 - May 2	● Prepare materials for student research celebration presentation and booth ● Present

Roles and Responsibilities:
Given that Farm Owl is a mixture of traditional engineering and software engineering (with a team of two), there are a couple ways the roles and responsibilities can be split up. One option is that one person handles all the physical and hardware side of the project, while the other works on the software side. This could allow for more consistency when it comes to the implementation. The second option could be intermixing responsibilities. One person works on the physical and hardware side for a sprint and then switches on the next. Both are viable options and have their benefits. We have decided to go with intermixing responsibilities since that should allow for a fresh set of eyes to locate and solve problems. Below are the responsibilities for each person per sprint.

Division of Labor:

Sprint 1	**Assigned Person**
• Hover-worthy airframe put together	**Cole**
• Wiring done	**Both**
• Basic elevation controls	**Mason**

Sprint 2	**Assigned Person**
• Forward, backward, left, right movement	**Cole**
• Camera driver functioning	**Both**
• Camera installed on airframe	**Mason**

Sprint 3	Assigned Person
• Start and complete image capturing	**Cole**
• Test flight driver	**Both**
• Start and complete image stitching	**Mason**

Sprint 4	Assigned Person
• Integrate image capturing and image stitching • Test camera capturing and stitching	**Both**

Sprint 5	Assigned Person
• Integrate drone driver • Drone-wide bug fixes and optimizations	**Both**

Sprint 6	Assigned Person
• Overall drone performance testing with flight and camera • Finishing touches	**Both**

Sprint 7	Assigned Person
• Prepare materials for student research celebration presentation and booth • Present	**Both**

Proposal Statement

Functional Requirements:

- The system must be able to hover in place
- The system must be able to fly ordinally
- The system must be autonomous when given valid input
- The system’s Raspberry Pi driver must control an image stitching algorithm, onboard camera, and motor drivers
- The system must have an algorithm to determine the optimal time/place to take a photo
- The system’s Raspberry Pi driver must control four DC motors at the same time
- The system must be able to store captured data for an arbitrary amount of time
- The system’s Raspberry Pi must be able to land with minimal damage to the system

Non-functional Requirements:

- The camera must be at least 1080p
- The entire system must cost less than $250
- The system must be able to fly without a controller

Performance Requirements:

- The image stitching algorithm should be >80% accurate
- The system should take 1 photo per 1000 square feet
- The system must be able to fly/hover for more than 10 minutes
- The system must weigh less than 0.55 pounds
- The system must be able to remain upright and stable in winds less than 2 mph

Interface Requirements:

- The Raspberry Pi must control the mechanical system (DC motors)
- The Raspberry Pi must control the onboard camera
- The Raspberry Pi should have a function to take and store a picture using the onboard camera
- The Raspberry Pi must activate the image stitcher once it has landed

Architecture:

Physical Frame
Hardware
Software

Tools/Standards:

Physical parts:

- Propellers
- 3D printed frame

Hardware:

- Raspberry Pi Zero
- Power Supply (Lipo battery)
- RGB Camera
- 4 DC Motors
- Electronic Speed Controller (ESC)
- Power Distribution Board (PSB)
- Flight Control Shield
- Soldering gun

Software:

- Python
- Docker
- Raspberry Pi OS
- OpenCV-Python

VCS/Agile:

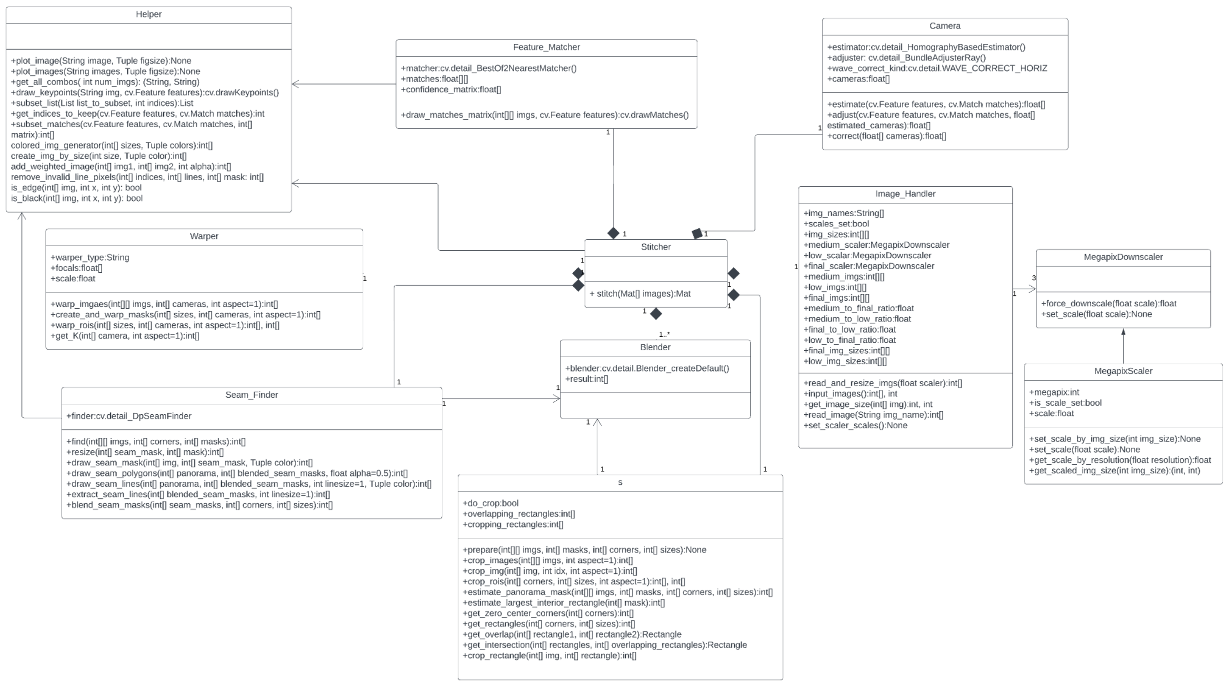
- Git
- Github
- Asana

Methodology

User Stories:

1) As a farmer, I want to see an aerial view of my farm land so that I can accurately fertilize my crops.
2) As a farmer, I want to easily fly a drone so that I do not cause harm to anything around me.
3) As a farmer, I can easily repair my drone so that I do not have to send it back to get repaired.

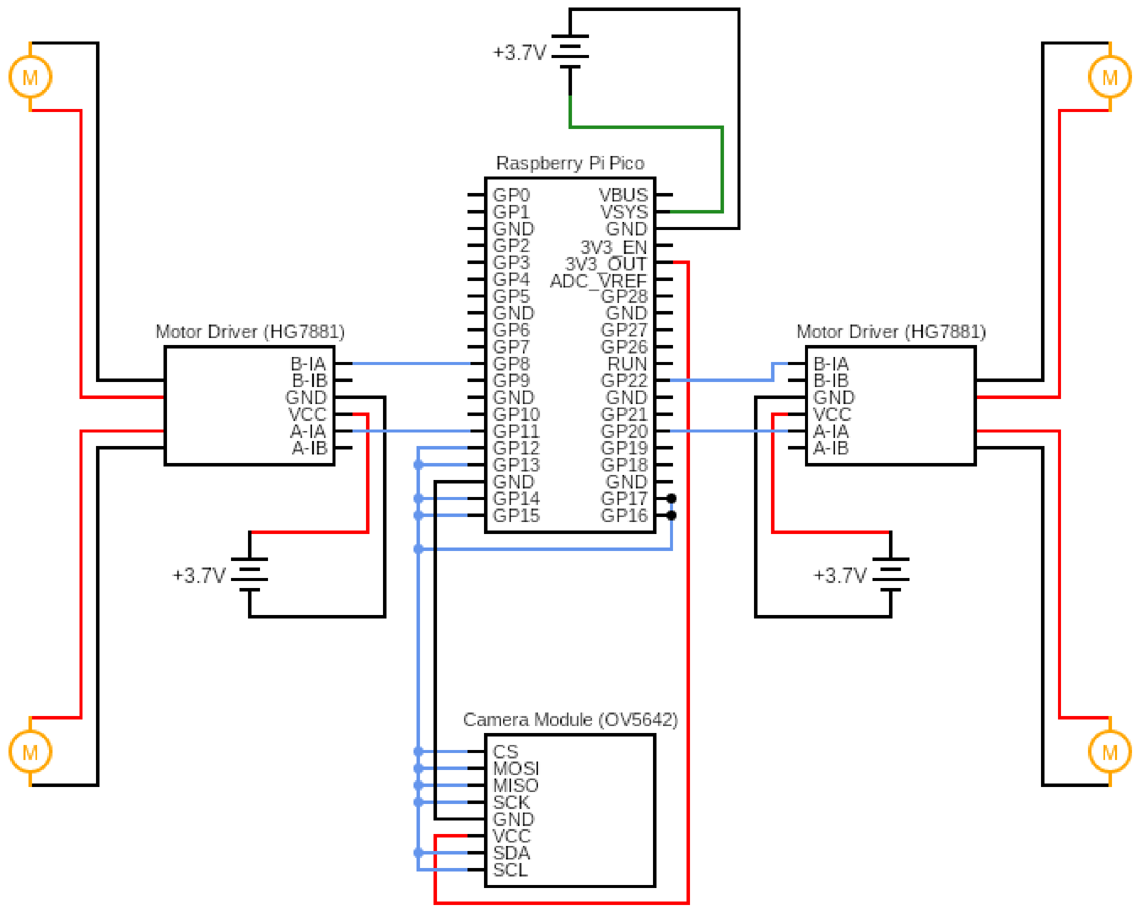
Image Stitcher



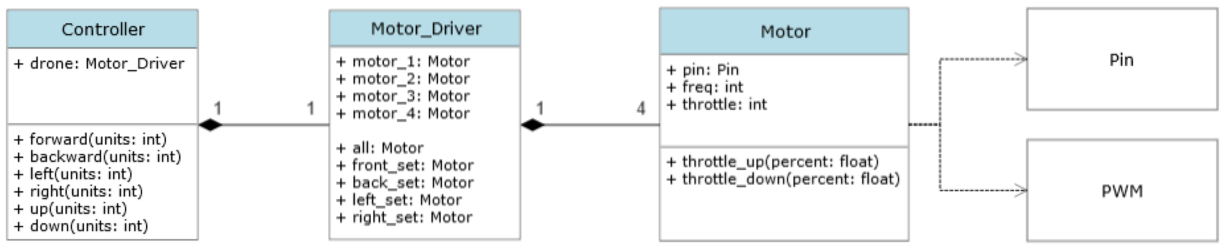
Helper
+plot_image(String image, Tuple figsize):None
+plot_images(String images, Tuple figsize):None
+get_all_combos(int num_imgs): (String, String)
+draw_keypoints(String img, cv.Feature features):cv.drawKeypoints()
+subset_list(List list_to_subset, int indices):List
+get_indices_to_keep(cv.Feature features, cv.Match matches):int
+subset_matches(cv.Feature features, cv.Match matches, int[] matrix):int[]
colored_img_generator(int[] sizes, Tuple colors):int[]
create_img_by_size(int size, Tuple color):int[]
add_weighted_image(int[] img1, int[] img2, int alpha):int[]
remove_invalid_line_pixels(int[] indices, int[] lines, int[] mask: int[]
is_edge(int[] img, int x, int y): bool
is_black(int[] img, int x, int y): bool
Feature_Matcher
+matcher:cv.detail_BestOf2NearestMatcher()
+matches:float[][]
+confidence_matrix:float[]
+draw_matches_matrix(int[][] imgs, cv.Feature features):cv.drawMatches()
Camera
+estimator:cv.detail_HomographyBasedEstimator()
+adjuster: cv.detail_BundleAdjusterRay()
+wave_correct_kind:cv.detail.WAVE_CORRECT_HORIZ
+cameras:float[]
+estimate(cv.Feature features, cv.Match matches):float[]
+adjust(cv.Feature features, cv.Match matches, float[] estimated_cameras):float[]
+correct(float[] cameras):float[]
Warper
+warper_type:String
+focals:float[]
+scale:float
+warp_imgaes(int[][] imgs, int[] cameras, int aspect=1):int[]
+create_and_warp_masks(int[] sizes, int[] cameras, int aspect=1):int[]
+warp_rois(int[] sizes, int[] cameras, int aspect=1):int[], int[]
+get_K(int[] camera, int aspect=1):int[]
Stitcher
+ stitch(Mat[] images):Mat
Image_Handler
+img_names:String[]
+scales_set:bool
+img_sizes:int[][]
+medium_scaler:MegapixDownscaler
+low_scalar:MegapixDownscaler
+final_scaler:MegapixDownscaler
+medium_imgs:int[][]
+low_imgs:int[][]
+final_imgs:int[][]
+medium_to_final_ratio:float
+medium_to_low_ratio:float
+final_to_low_ratio:float
+low_to_final_ratio:float
+final_img_sizes:int[][]
+low_img_sizes:int[][]
+read_and_resize_imgs(float scaler):int[]
+input_images():int[], int
+get_image_size(int[] img):int, int
+read_image(String img_name):int[]
+set_scaler_scales():None
MegapixDownscaler
+force_downscale(float scale):float
+set_scale(float scale):None
MegapixScaler
+megapix:int
+is_scale_set:bool
+scale:float
+set_scale_by_img_size(int img_size):None
+set_scale(float scale):None
+get_scale_by_resolution(float resolution):float
+get_scaled_img_size(int img_size):(int, int)
Blender
+blender:cv.detail.Blender_createDefault()
+result:int[]
Seam_Finder
+finder:cv.detail_DpSeamFinder
+find(int[][] imgs, int[] corners, int[] masks):int[]
+resize(int[] seam_mask, int[] mask):int[]
+draw_seam_mask(int[] img, int[] seam_mask, Tuple color):int[]
+draw_seam_polygons(int[] panorama, int[] blended_seam_masks, float alpha=0.5):int[]
+draw_seam_lines(int[] panorama, int[] blended_seam_masks, int linesize=1, Tuple color):int[]
+extract_seam_lines(int[] blended_seam_masks, int linesize=1):int[]
+blend_seam_masks(int[] seam_masks, int[] corners, int[] sizes):int[]
S
+do_crop:bool
+overlapping_rectangles:int[]
+cropping_rectangles:int[]
+prepare(int[][] imgs, int[] masks, int[] corners, int[] sizes):None
+crop_images(int[][] imgs, int aspect=1):int[]
+crop_img(int[] img, int idx, int aspect=1):int[]
+crop_rois(int[] corners, int[] sizes, int aspect=1):int[], int[]
+estimate_panorama_mask(int[][] imgs, int[] masks, int[] corners, int[] sizes):int[]
+estimate_largest_interior_rectangle(int[] mask):int[]
+get_zero_center_corners(int[] corners):int[]
+get_rectangles(int[] corners, int[] sizes):int[]
+get_overlap(int[] rectangle1, int[] rectangle2):Rectangle
+get_intersection(int[] rectangles, int[] overlapping_rectangles):Rectangle
+crop_rectangle(int[] img, int[] rectangle):int[]
1
1..*
3

The image stitching algorithm will use images and the metadata from them to combine the images into one single picture with a high field of view. The image memory class will pull all images taken from the flight into memory and store them for use by the intermediate model. The intermediate model is a one image per instance class. Each instance will take sequential images (initiated by the Image_Compute class) and calculate their proper location. They will then be combined for the end product. The other two classes are dedicated to computing the result and saving it to memory. The ending image will be stored somewhere easily accessible (i.e. plugging in a USB drive) for use/collection.

Wire Diagram :



Drone Class Diagram:



Mediator Pattern:
The mediator pattern was chosen to control the image stitching process of the project. Due to the multi-faceted process of stitching together an image, many different processes must be channeled and controlled together via one source; we'll call that the stitcher. The stitcher mediates between, or invokes all other classes in a directed and acyclic fashion. One could consider the stitcher in a similar fashion to data warehousing concepts—a DAG, or Directed Acyclic Graph. The stitcher must invoke each class, as each class is dependent upon another; the process builds upon itself. This way, no tightly coupled process can get too far ahead of itself, or try to start without necessary data. The image warping algorithm (via camera estimation) cannot warp without a camera object! The mediator pattern allows us to keep neat order of the classes we instantiate before another.

Design tradeoffs:
1. The Raspberry Pi Zero has been sold out for months. Scalpers and sharks on the internet are selling them for hundreds of dollars. This makes it incredibly difficult to get a (usually $5) microcomputer. The two biggest reasons the Zero was chosen was for its small weight, and camera ribbon cable. No other small Raspberry Pis have an adequate choice for camera plugins. The Zero is marketed as the go-to choice for "media projects". However, with the short supply, we will likely choose the Raspberry Pi Pico. This computer is actually smaller and cheaper than the Zero, but much less powerful. The lack of a dedicated camera plugin will also make capturing pictures much more difficult. Thankfully, we will take "difficult but possible" over "impossible" any day.

2. Upon initial investigation, we realized that the microcomputer we were using does not have enough memory to store the image stitching algorithm, let alone a single JPEG! Thus we determined to code the algorithm and store it separately, with a potential use case of putting it in the cloud. Although the new workflow is out of scope for this project, the new process would look like this: Drone flies over land and surveys, streaming data into the Cloud via a messaging subscription (like Google Pub/Sub); the drone then eventually lands and sends a 'finished' message to the same service. The cloud project

would then proceed to run a function holding the stitcher code on the images, dropped into a bucket via the Pub/Sub chain mentioned earlier. From there, the project could run another function to drop or email the stitched image into another bucket or someone's email via another subscription chain. This would skirt the physical limitations of the micro-drone, and create a scalable solution for anyone using it.

3. Another design tradeoff is the size of the drone itself. Drones with a larger size tend to be more stable than smaller drones. This is great for general flight performance, however performance from size comes at the cost of added weight. Farm Owl needs to be stable enough to hover in place and fly, but also weigh less than 250 grams to avoid needing to register the drone. In other words, stability should be maximized while size and weight should be minimized. At this point in time, coming to a conclusive decision on the size of the drone would limit our options for performance. We are going to experiment with drone frame sizes and see what best fits the project.

Results

Drone:

Creating a drone from scratch is no easy feat. There are usually many disciplines involved (mechanical engineer, electrical engineer, computer engineer/computer science, ect), and even more intricacies to coincide with them. Farm Owl did not have these other areas of expertise and that was a significant problem. Computer scientists are not known for their prowess in circuitry or CAD drawings, and require an immense amount of studying in order to create a project like this.

Because of the subtleties and complexities of more physical fields, the drone portion of Farm Owl was definitely ambitious. Especially when considering that this project was created by students in many other demanding courses throughout the semester. Many of the initially desired features are not in a working state and were above what should have been expected for this project. The drone portion may have been more achievable if this was a full time job, but obviously that is not achievable for full time students.

Image Stitcher:

Creating panoramic images from separate source images is more complicated than it sounds on paper. There is much multivariable calculus involved: gradients, partial derivatives, etc. On the other hand however, the process is quite well documented.[19] OpenCV contains all the tools needed to create a full, accurate and relatively fast stitching algorithm.

The actual resulting code utilizes several built in modules from OpenCV, as well as classes and functions built by individual academic contributors. The process of stitching an image (as outlined above in ‘Background’) is complicated and requires many moving parts and estimations, but ultimately this makes the whole process easier. It’s not one, large jumble of code; but instead many small pieces working together, such as a camera estimator performing an estimation, and a cropping algorithm cropping an image using that estimation.

[19] Matthew Brown, David Lowe: Automatic Panoramic Image Stitching

The entire working process took much longer than expected to build this algorithm, but the result was high quality. This high quality was due to the fact of several extraordinarily helpful resources along the way, since this process is so heavily documented.

Code Appendix

Drone:

Controller:

```python
from code.pi.motor_driver import Motor_Driver as MD


class Controller:

   @staticmethod
   def up(percent, increment: int = 5):
       for m in MD.all:
           m.throttle_up(percent, increment)

   @staticmethod
   def down(percent, decrement: int = 5):
       for m in MD.all:
           m.throttle_down(percent, decrement)

   @staticmethod
   def forward(percent, decrement: int = 5):
       for m in MD.front_set:
           m.throttle_down(percent, decrement)

   @staticmethod
   def backward(percent, decrement: int = 5):
       for m in MD.back_set:
           m.throttle_down(percent, decrement)

   @staticmethod
   def left(percent, decrement: int = 5):
       for m in MD.left_set:
           m.throttle_down(percent, decrement)

   @staticmethod
   def right(percent, decrement: int = 5):
       for m in MD.right_set:
           m.throttle_down(percent, decrement)
```

Motor Driver:

```python
from code.pi.motor import Motor


class Motor_Driver:
   motor_1 = Motor(8)
   motor_2 = Motor(22)
   motor_3 = Motor(11)
   motor_4 = Motor(20)

   # Group motors together into sets
   all = (motor_1, motor_2, motor_3, motor_4)
   front_set = (motor_1, motor_2)
   back_set = (motor_3, motor_4)
   left_set = (motor_1, motor_3)
   right_set = (motor_2, motor_4)
```

Motor:

```python
from machine import Pin, PWM
import utime


class Motor:

   def __init__(self, pin_num: int, freq: int = 1000):
       self.motor = PWM(Pin(pin_num))
       self.motor.freq(freq)
       self.throttle = 0

   def throttle_up(self, percent, increment: int):
       for t in range(0, 65536 * percent, increment):
           self.throttle = t
           self.motor.duty_u16(self.throttle)
           print(self.throttle)
           utime.sleep_ms(5)

   def throttle_down(self, percent, decrement: int):
       for t in range(0, 65536 * percent, -decrement):
           self.throttle = t
           self.motor.duty_u16(self.throttle)
           print(self.throttle)
           utime.sleep_ms(5)
```

Image Stitcher:

Image Handler:

```
from megapix_scaler import MegapixDownscaler
import cv2 as cv
class Image_Handler:
    def __init__(self, imgs):
        self.img_names = imgs
        self.scales_set = False
        self.img_sizes = []
        # scaler init (to actually scale an image proportionately
        self.medium_scaler = MegapixDownscaler(0.6)
        self.low_scaler = MegapixDownscaler(0.1)
        self.final_scaler = MegapixDownscaler(-1)
        # use above scalers to scale images down to respective sizes
        self.medium_imgs = list(self.read_and_resize_imgs(self.medium_scaler))
        self.low_imgs = list(self.read_and_resize_imgs(self.low_scaler))
        self.final_imgs = list(self.read_and_resize_imgs(self.final_scaler))
        # ratios (for cameras and croppers)
        self.medium_to_final_ratio = self.final_scaler.scale /
self.medium_scaler.scale
        self.medium_to_low_ratio = self.low_scaler.scale /
self.medium_scaler.scale
        self.final_to_low_ratio = self.low_scaler.scale /
self.final_scaler.scale
        self.low_to_final_ratio = self.final_scaler.scale /
self.low_scaler.scale

        # sizes (for warpers)
        self.final_img_sizes = [self.final_scaler.get_scaled_img_size(sz) for
sz in self.img_sizes]
        self.low_img_sizes = [self.low_scaler.get_scaled_img_size(sz) for sz in
self.img_sizes]
        # medium size is get_img_sizes() or img[x].shape
    def read_and_resize_imgs(self, scaler):
        for img, size in self.input_images():
            desired_size = scaler.get_scaled_img_size(size)
            yield cv.resize(img, desired_size,
interpolation=cv.INTER_LINEAR_EXACT)
    def resize_imgs_by_scaler(self, imgs, scaler):
        for img, size in zip(imgs, self.img_sizes):
            yield self.resize_img_by_scaler(scaler, size, img)
    def input_images(self):
        self.img_sizes = []
        for name in self.img_names:
            img = self.read_image(name)
            size = self.get_image_size(img)
            self.img_sizes.append(size)
            self.set_scaler_scales()
            yield img, size
    @staticmethod
    def get_image_size(img):
        # width, height (everything in OpenCV is backwards)
        return img.shape[1], img.shape[0]
    @staticmethod
    def read_image(img_name):
        img = cv.imread(img_name)
        return img
    def set_scaler_scales(self):
        if not self.scales_set:
            first_img_size = self.img_sizes[0]
            self.medium_scaler.set_scale_by_img_size(first_img_size)
            self.low_scaler.set_scale_by_img_size(first_img_size)
            self.final_scaler.set_scale_by_img_size(first_img_size)
        self.scales_set = True
```

Feature Matcher:

```
from helper import get_all_combos
import math
import cv2 as cv
import numpy as np
class Feature_Matcher:
    def __init__(self, features):
        self.matcher = cv.detail_BestOf2NearestMatcher()
        self.matches = self.matcher.apply2(features)
        self.matcher.collectGarbage()
        # match matrix
        matrix_dimension = int(math.sqrt(len(self.matches)))
        rows = []
        for i in range(0, len(self.matches), matrix_dimension):
            rows.append(self.matches[i: i + matrix_dimension])
        self.match_matrix = np.array(rows)
        # confidence matrix (confidence from match matrix)
        self.confidence_matrix = np.array([[m.confidence for m in row] for row
in self.match_matrix])
    def draw_matches_matrix(self, imgs, features, **kwargs):
        matches_matrix = self.match_matrix
        for idx1, idx2 in get_all_combos(len(imgs)):
            match = matches_matrix[idx1, idx2]
            if match.confidence < 1:
                continue
            kwargs["matchesMask"] = match.getInliers()
            kwargs.setdefault("flags",
cv.DrawMatchesFlags_NOT_DRAW_SINGLE_POINTS)
            yield idx1, idx2, cv.drawMatches(
                imgs[idx1],
                features[idx1].getKeypoints(),
               imgs[idx2],
               features[idx2].getKeypoints(),
               match.getMatches(),
               None,
               **kwargs
           )
```

Camera Estimator:

```
import cv2 as cv
import numpy as np
class Camera:
   def __init__(self, features, matches, **kwargs):
       self.estimator = cv.detail_HomographyBasedEstimator(**kwargs)
       self.adjuster = cv.detail_BundleAdjusterRay()
       self.wave_correct_kind = cv.detail.WAVE_CORRECT_HORIZ
       self.adjuster.setRefinementMask(np.ones((3, 3), np.uint8))
       self.adjuster.setConfThresh(1.0)
       self.cameras = self.estimate(features, matches)
       self.cameras = self.adjust(features, matches, self.cameras)
       self.cameras = self.correct(self.cameras)
   def estimate(self, features, pairwise_matches):
       b, cameras = self.estimator.apply(features, pairwise_matches, None)
       for cam in cameras:
           cam.R = cam.R.astype(np.float32)
       return cameras
   def adjust(self, features, pairwise_matches, estimated_cameras):
       b, cameras = self.adjuster.apply(features, pairwise_matches,
estimated_cameras)
       return cameras
   def correct(self, cameras):
       rmats = [np.copy(cam.R) for cam in cameras]
       rmats = cv.detail.waveCorrect(rmats, self.wave_correct_kind)
       for idx, cam in enumerate(cameras):
           cam.R = rmats[idx]
       return cameras
```

Warper:

```
from statistics import median
import cv2 as cv
import numpy as np
class Warper:
   def __init__(self, cameras):
       self.warper_type = 'spherical'
       focals = [cam.focal for cam in cameras]
       self.scale = median(focals)
   def warp_images(self, imgs, cameras, aspect=1):
       for img, camera in zip(imgs, cameras):
           warper = cv.PyRotationWarper(self.warper_type, self.scale * aspect)
```

```python
            _, warped_image = warper.warp(
                img,
                Warper.get_K(camera, aspect),
                camera.R,
                cv.INTER_LINEAR,
                cv.BORDER_REFLECT,
            )
            yield warped_image
    def create_and_warp_masks(self, sizes, cameras, aspect=1):
        for size, camera in zip(sizes, cameras):
            warper = cv.PyRotationWarper(self.warper_type, self.scale * aspect)
            mask = 255 * np.ones((size[1], size[0]), np.uint8)
            _, warped_mask = warper.warp(
                mask,
                Warper.get_K(camera, aspect),
                camera.R,
                cv.INTER_NEAREST,
                cv.BORDER_CONSTANT,
            )
            yield warped_mask
    def warp_rois(self, sizes, cameras, aspect=1):
        roi_corners = []
        roi_sizes = []
        for size, camera in zip(sizes, cameras):
            warper = cv.PyRotationWarper(self.warper_type, self.scale * aspect)
            K = Warper.get_K(camera, aspect)
            roi = warper.warpRoi(size, K, camera.R)
            roi_corners.append(roi[0:2])
            roi_sizes.append(roi[2:4])
        return roi_corners, roi_sizes
    @staticmethod
    def get_K(camera, aspect=1):
        K = camera.K().astype(np.float32)
        """ Modification of intrinsic parameters needed if cameras were
        obtained on different scale than the scale of the Images which should
        be warped """
        K[0, 0] *= aspect
        K[0, 2] *= aspect
        K[1, 1] *= aspect
        K[1, 2] *= aspect
        return K
```

Cropper:

```python
from collections import namedtuple
import cv2 as cv
from blender import Blender
import largestinteriorrectangle
# code from : LukasAlexanderWeber
```

```python
class Rectangle(namedtuple("Rectangle", "x y width height")):
   __slots__ = ()
   @property
   def area(self):
       return self.width * self.height
   @property
   def corner(self):
       return (self.x, self.y)
   @property
   def size(self):
       return (self.width, self.height)
   @property
   def x2(self):
       return self.x + self.width
   @property
   def y2(self):
       return self.y + self.height
   def times(self, x):
       return Rectangle(*(int(round(i * x)) for i in self))
   def draw_on(self, img, color=(0, 0, 255), size=1):
       if len(img.shape) == 2:
           img = cv.cvtColor(img, cv.COLOR_GRAY2RGB)
       start_point = (self.x, self.y)
       end_point = (self.x2 - 1, self.y2 - 1)
       cv.rectangle(img, start_point, end_point, color, size)
       return img
class Cropper:
   DEFAULT_CROP = True
   def __init__(self, crop=DEFAULT_CROP):
       self.do_crop = crop
       self.overlapping_rectangles = []
       self.cropping_rectangles = []
   def prepare(self, imgs, masks, corners, sizes):
       if self.do_crop:
           mask = self.estimate_panorama_mask(imgs, masks, corners, sizes)
           lir = self.estimate_largest_interior_rectangle(mask)
           corners = self.get_zero_center_corners(corners)
           rectangles = self.get_rectangles(corners, sizes)
           self.overlapping_rectangles = self.get_overlaps(rectangles, lir)
           self.intersection_rectangles = self.get_intersections(
               rectangles, self.overlapping_rectangles
           )
   def crop_images(self, imgs, aspect=1):
       for idx, img in enumerate(imgs):
           yield self.crop_img(img, idx, aspect)
   def crop_img(self, img, idx, aspect=1):
       if self.do_crop:
           intersection_rect = self.intersection_rectangles[idx]
           scaled_intersection_rect = intersection_rect.times(aspect)
```

```
            cropped_img = self.crop_rectangle(img, scaled_intersection_rect)
            return cropped_img
        return img
    def crop_rois(self, corners, sizes, aspect=1):
        if self.do_crop:
            scaled_overlaps = [r.times(aspect) for r in
self.overlapping_rectangles]
            cropped_corners = [r.corner for r in scaled_overlaps]
            cropped_corners = self.get_zero_center_corners(cropped_corners)
            cropped_sizes = [r.size for r in scaled_overlaps]
            return cropped_corners, cropped_sizes
        return corners, sizes
    @staticmethod
    def estimate_panorama_mask(imgs, masks, corners, sizes):
        mask = Blender(imgs, masks, corners, sizes).result_mask
        return mask
    def estimate_largest_interior_rectangle(self, mask):
        # largestinteriorrectangle is only imported if cropping
        # is explicitly desired (needs some time to compile at the first run!)
        contours, hierarchy = cv.findContours(mask, cv.RETR_TREE,
cv.CHAIN_APPROX_NONE)
        contour = contours[0][:, 0, :]
        lir = largestinteriorrectangle.lir(mask > 0, contour)
        lir = Rectangle(*lir)
        return lir
    @staticmethod
    def get_zero_center_corners(corners):
        min_corner_x = min([corner[0] for corner in corners])
        min_corner_y = min([corner[1] for corner in corners])
        return [(x - min_corner_x, y - min_corner_y) for x, y in corners]
    @staticmethod
    def get_rectangles(corners, sizes):
        rectangles = []
        for corner, size in zip(corners, sizes):
            rectangle = Rectangle(*corner, *size)
            rectangles.append(rectangle)
        return rectangles
    @staticmethod
    def get_overlaps(rectangles, lir):
        return [Cropper.get_overlap(r, lir) for r in rectangles]
    @staticmethod
    def get_overlap(rectangle1, rectangle2):
        x1 = max(rectangle1.x, rectangle2.x)
        y1 = max(rectangle1.y, rectangle2.y)
        x2 = min(rectangle1.x2, rectangle2.x2)
        y2 = min(rectangle1.y2, rectangle2.y2)
        return Rectangle(x1, y1, x2 - x1, y2 - y1)
    @staticmethod
    def get_intersections(rectangles, overlapping_rectangles):
       return [
           Cropper.get_intersection(r, overlap_r)
           for r, overlap_r in zip(rectangles, overlapping_rectangles)
       ]
   @staticmethod
   def get_intersection(rectangle, overlapping_rectangle):
       x = abs(overlapping_rectangle.x - rectangle.x)
       y = abs(overlapping_rectangle.y - rectangle.y)
       width = overlapping_rectangle.width
       height = overlapping_rectangle.height
       return Rectangle(x, y, width, height)
   @staticmethod
   def crop_rectangle(img, rectangle):
       return img[rectangle.y: rectangle.y2, rectangle.x: rectangle.x2]
```

Seam Finder:

```
import cv2 as cv
import numpy as np
from helper import add_weighted_image, remove_invalid_line_pixels,
colored_img_generator
from blender import Blender
class Seam_Finder:
   def __init__(self):
       self.finder = cv.detail_DpSeamFinder("COLOR")
   def find(self, imgs, corners, masks):
       imgs_float = [img.astype(np.float32) for img in imgs]
       return self.finder.find(imgs_float, corners, masks)
   @staticmethod
   def resize(seam_mask, mask):
       dilated_mask = cv.dilate(seam_mask, None)
       resized_seam_mask = cv.resize(
           dilated_mask, (mask.shape[1], mask.shape[0]), 0, 0,
cv.INTER_LINEAR_EXACT
       )
       return cv.bitwise_and(resized_seam_mask, mask)
   @staticmethod
   def draw_seam_mask(img, seam_mask, color=(0, 0, 0)):
       seam_mask = cv.UMat.get(seam_mask)
       overlaid_img = np.copy(img)
       overlaid_img[seam_mask == 0] = color
       return overlaid_img
   @staticmethod
   def draw_seam_polygons(panorama, blended_seam_masks, alpha=0.5):
       return add_weighted_image(panorama, blended_seam_masks, alpha)
   @staticmethod
   def draw_seam_lines(panorama, blended_seam_masks, linesize=1, color=(0, 0,
255)):
        seam_lines = Seam_Finder.extract_seam_lines(blended_seam_masks,
linesize)
        panorama_with_seam_lines = panorama.copy()
        panorama_with_seam_lines[seam_lines == 255] = color
        return panorama_with_seam_lines
    @staticmethod
    def extract_seam_lines(blended_seam_masks, linesize=1):
        seam_lines = cv.Canny(np.uint8(blended_seam_masks), 100, 200)
        seam_indices = (seam_lines == 255).nonzero()
        seam_lines = remove_invalid_line_pixels(
            seam_indices, seam_lines, blended_seam_masks
        )
        kernelsize = linesize + linesize - 1
        kernel = np.ones((kernelsize, kernelsize), np.uint8)
        return cv.dilate(seam_lines, kernel)
    @staticmethod
    def blend_seam_masks(seam_masks, corners, sizes):
        imgs = colored_img_generator(sizes)
        blended_seam_masks, _ = Blender(
            imgs, seam_masks, corners, sizes
        ).result
        return blended_seam_masks
```

Blender:

```
import cv2 as cv
import numpy as np
class Blender:
    def __init__(self, imgs, masks, corners, sizes):
        # init blender
        self.blender = cv.detail.Blender_createDefault(cv.detail.Blender_NO)
        # prepare step
        roi = cv.detail.resultRoi(corners=corners, sizes=sizes)
        self.blender.prepare(roi)
        # feed to blender
        for img, mask, corner in zip(imgs, masks, corners):
            self.blender.feed(cv.UMat(img.astype(np.int16)), mask, corner)
        # blend and result
        self.result, self.result_mask = None, None
        self.result, self.result_mask = self.blender.blend(self.result,
self.result_mask)
        self.result = cv.convertScaleAbs(self.result)
```

Helper:

```
import matplotlib.pyplot as plt
import cv2 as cv
import numpy as np
import random
from itertools import chain
```

```python
def plot_image(img, figsize_in_inches=(5, 5)):
    fig, ax = plt.subplots(figsize=figsize_in_inches)
    ax.imshow(cv.cvtColor(img, cv.COLOR_BGR2RGB))
    plt.show()
def plot_images(imgs, figsize_in_inches=(5, 5)):
    fig, axs = plt.subplots(1, len(imgs), figsize=figsize_in_inches)
    for col, img in enumerate(imgs):
        axs[col].imshow(cv.cvtColor(img, cv.COLOR_BGR2RGB))
    plt.show()
def get_all_combos(num_imgs):
    ii, jj = np.triu_indices(num_imgs, k=1)
    for i, j in zip(ii, jj):
        yield i, j
def draw_keypoints(img, features, **kwargs):
    kwargs.setdefault("color", (random.randint(0, 200), random.randint(100,
255), random.randint(50, 150)))
    keypoints = features.getKeypoints()
    return cv.drawKeypoints(img, keypoints, None, **kwargs)
def subset_list(list_to_subset, indices):
    return [list_to_subset[i] for i in indices]
def get_indices_to_keep(features, pairwise_matches):
    indices = cv.detail.leaveBiggestComponent(features, pairwise_matches, 1)
    return indices
def subset_matches(features, matches, match_matrix):
    indices = cv.detail.leaveBiggestComponent(features, matches, 1)
    matches_matrix = match_matrix
    matches_matrix_subset = matches_matrix[np.ix_(indices, indices)]
    matches_subset_list =
list(chain.from_iterable(matches_matrix_subset.tolist()))
    return matches_subset_list
def colored_img_generator(
        sizes,
        colors=(
            (255, 000, 000),  # Blue
            (000, 000, 255),  # Red
            (000, 255, 000),  # Green
            (000, 255, 255),  # Yellow
            (255, 000, 255),  # Magenta
            (128, 128, 255),  # Pink
            (128, 128, 128),  # Gray
            (000, 000, 128),  # Brown
            (000, 128, 255), # Orange
        ),
):
    for idx, size in enumerate(sizes):
        if idx + 1 > len(colors):
            raise ValueError(
                "Not enough default colors! Pass additional "
                'colors to "colors" parameter'
            )
        yield create_img_by_size(size, colors[idx])
def create_img_by_size(size, color=(0, 0, 0)):
    width, height = size
    img = np.zeros((height, width, 3), np.uint8)
    img[:] = color
    return img
def add_weighted_image(img1, img2, alpha):
    return cv.addWeighted(img1, alpha, img2, (1.0 - alpha), 0.0)
def remove_invalid_line_pixels(indices, lines, mask):
    for x, y in zip(*indices):
        if is_edge(mask, x, y):
            lines[x, y] = 0
    return lines
def is_edge(img, x, y):
    return any([
        is_black(img, x, y),
        is_black(img, x + 1, y),
        is_black(img, x - 1, y),
        is_black(img, x, y + 1),
        is_black(img, x, y - 1),
    ])
def is_black(img, x, y):
    return np.all(img[x, y] == 0)
```

Stitcher:

```
from helper import plot_images, plot_image, draw_keypoints, subset_list,
get_indices_to_keep, subset_matches
from image_handler import Image_Handler
from camera import Camera
import cv2 as cv
from blender import Blender
from feature_matcher import Feature_Matcher
from cropper import Cropper
from warper import Warper
from seam_finder import Seam_Finder
class Stitcher:
    @staticmethod
    def stitch(imgs):
        image_list = imgs
        img_handler = Image_Handler(image_list)
        medium_imgs = img_handler.medium_imgs
        low_imgs = img_handler.low_imgs
        final_imgs = img_handler.final_imgs
        plot_images(low_imgs, (20, 20))
        """https://docs.opencv.org/4.x/d0/d13/classcv_1_1Feature2D.html"""
        feature_detector = cv.ORB.create()
        features = [cv.detail.computeImageFeatures2(feature_detector, img) for
img in medium_imgs]
        keypoints_center_img = draw_keypoints(medium_imgs[1], features[1])
        plot_image(keypoints_center_img, (15, 10))
        matcher = Feature_Matcher(features)
        matches = matcher.matches
        all_relevant_matches = matcher.draw_matches_matrix(medium_imgs,
features, matchColor=(255, 0, 0))
        for idx1, idx2, img in all_relevant_matches:
            print(f"Matches Image {idx1 + 1} to Image {idx2 + 1}")
            plot_image(img, (20, 10))
        indices = get_indices_to_keep(features, matches)
        medium_imgs = subset_list(medium_imgs, indices)
        low_imgs = subset_list(low_imgs, indices)
        final_imgs = subset_list(final_imgs, indices)
        features = subset_list(features, indices)
        matches = subset_matches(features, matches, matcher.match_matrix)
        img_names = subset_list(img_handler.img_names, indices)
        img_sizes = subset_list(img_handler.img_sizes, indices)
        img_handler.img_names, img_handler.img_sizes = img_names, img_sizes
        print(img_handler.img_names)
        print(matcher.confidence_matrix)
        camera = Camera(features, matches)
        cameras = camera.cameras
        warper = Warper(cameras)
        low_sizes = img_handler.low_img_sizes
        camera_aspect = img_handler.medium_to_low_ratio  # since cameras were
obtained on medium imgs
        warped_low_imgs = list(warper.warp_images(low_imgs, cameras,
camera_aspect))
        warped_low_masks = list(warper.create_and_warp_masks(low_sizes,
cameras, camera_aspect))
        low_corners, low_sizes = warper.warp_rois(low_sizes, cameras,
camera_aspect)
        final_sizes = img_handler.final_img_sizes
        camera_aspect = img_handler.medium_to_final_ratio  # since cameras were
obtained on medium imgs
        warped_final_imgs = list(warper.warp_images(final_imgs, cameras,
camera_aspect))
        warped_final_masks = list(warper.create_and_warp_masks(final_sizes,
cameras, camera_aspect))
```

```
       final_corners, final_sizes = warper.warp_rois(final_sizes, cameras,
camera_aspect)
       plot_images(warped_low_imgs, (10, 10))
       plot_images(warped_low_masks, (10, 10))
       cropper = Cropper()
       mask = cropper.estimate_panorama_mask(warped_low_imgs,
warped_low_masks, low_corners, low_sizes)
       plot_image(mask, (5, 5))
       lir = cropper.estimate_largest_interior_rectangle(mask)
       plot = lir.draw_on(mask, size=2)
       plot_image(plot, (5, 5))
       low_corners = cropper.get_zero_center_corners(low_corners)
       rectangles = cropper.get_rectangles(low_corners, low_sizes)
       plot = rectangles[1].draw_on(plot, (0, 255, 0), 2)  # The rectangle of
the center img
       plot_image(plot, (5, 5))
       overlap = cropper.get_overlap(rectangles[1], lir)
       plot = overlap.draw_on(plot, (255, 0, 0), 2)
       plot_image(plot, (5, 5))
       intersection = cropper.get_intersection(rectangles[1], overlap)
       plot = intersection.draw_on(warped_low_masks[1], (255, 0, 0), 2)
       plot_image(plot, (2.5, 2.5))
       cropper.prepare(warped_low_imgs, warped_low_masks, low_corners,
low_sizes)
       cropped_low_masks = list(cropper.crop_images(warped_low_masks))
       cropped_low_imgs = list(cropper.crop_images(warped_low_imgs))
       low_corners, low_sizes = cropper.crop_rois(low_corners, low_sizes)
       lir_aspect = img_handler.low_to_final_ratio  # since lir was obtained
on low imgs
       cropped_final_masks = list(cropper.crop_images(warped_final_masks,
lir_aspect))
       cropped_final_imgs = list(cropper.crop_images(warped_final_imgs,
lir_aspect))
       final_corners, final_sizes = cropper.crop_rois(final_corners,
final_sizes, lir_aspect)
       seam_finder = Seam_Finder()
       seam_masks = seam_finder.find(cropped_low_imgs, low_corners,
cropped_low_masks)
       seam_masks = [seam_finder.resize(seam_mask, mask) for seam_mask, mask
in zip(seam_masks, cropped_final_masks)]
       seam_masks_plots = [Seam_Finder.draw_seam_mask(img, seam_mask) for img,
seam_mask in
```

```
                                   zip(cropped_final_imgs, seam_masks)]
        plot_images(seam_masks_plots, (15, 10))
        panorama = Blender(cropped_final_imgs, seam_masks, final_corners,
final_sizes).result
        plot_image(panorama, (20, 20))
imgs = [‘x1’,’x2’,’x3’]
Stitcher.stitch(imgs)
```

Citations

1. https://www.ag.ndsu.edu/publications/crops/agricultural-remote-sensing-basics
2. https://censystech.com/agriculture/
3. https://www.federaldroneregistration.com/faq
4. https://education.nationalgeographic.org/resource/dead-zone
5. https://oceantoday.noaa.gov/deadzonegulf-2021/welcome.html
6. https://www.raspberrypi.com/products/raspberry-pi-zero/
7. https://docs.opencv.org/4.x/d0/de3/tutorial_py_intro.html
8. https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=413336&tag=1
9. https://www.ibm.com/cloud/learn/unsupervised-learning
10. https://arxiv.org/abs/2202.01891
11. https://stanford.edu/~cpiech/cs221/handouts/kmeans.html
12. https://www.ibm.com/cloud/learn/supervised-learning
13. https://www.simplilearn.com/tutorials/deep-learning-tutorial/perceptron
14. https://acloudguru.com/course/introduction-to-machine-learning
15. https://www.ibm.com/cloud/learn/convolutional-neural-networks
16. https://opt-ml.org/papers/2021/paper53.pdf
17. https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e
18. https://machinelearningmastery.com/tour-of-ensemble-learning-algorithms/
19. http://matthewalunbrown.com/papers/ijcv2007.pdf