Team Skeyes

Dr. Abolfazl Razi

Final Report and User Manual
16 April 2021

Mohammed Almershed msa379@nau.edu
Daniel Copley djc449@nau.edu
Noah Levie nwl5@nau.edu

EE486C Capstone Design
Team 2
Skeyes Structural Monitoring Drone
16 April 2021

Table of Contents

EE486C Capstone Design
Team 2
Skeyes Structural Monitoring Drone
16 April 2021

Introduction

Client Background

Our group, Team Skeyes, is working with Dr. Abolfazl Razi, an engineering professor at Northern Arizona University in the Wireless Networking and Smart Health (WiNeSH) research laboratory. His personal projects are centered around predictive modeling for different applications including Wireless Networking, Smart Cities, IoT and Aerial Systems. His goal is to design new machine learning tools that model and predict network status change, user behavioral trends, and traffic mobility in order to accommodate predictable events by making early decisions. He also does work in the medical engineering field, involving the development of tools for predictive modeling of biomedical signals for smart health applications. All of his projects are supported by the NSF, NIH U54, US Airforce Research Laboratory, and Arizona Board of Regents (ABOR). A link to his personal website and portfolio can be found on the NAU website at https://www.cefns.nau.edu/~ar2843/.

Problem Statement

Conventional approaches to building inspection are laborious, costly, and dangerous. As it stands, an inspector has to personally travel around any building he would like to inspect; furthermore, in the case of a particularly tall building, he would have to be elevated to thoroughly survey the external surfaces of the structure. This process is slow, expensive, and potentially dangerous. On the contrary, the use of an automated drone for such a task would be faster, cheaper, safer, and more convenient for everyone involved. The drone will allow a user to plug a device into their laptop and give the program some basic instructions, then receive a streamlined output of video data that they can use in lieu of performing a manual inspection. The user will not have to physically move in order to investigate the building for flaws, they can stay in one place and observe the process from a comfortable or convenient location. This drone additionally makes the inspection of taller buildings more feasible, as the operator will not have to elevate themselves in order to achieve a close investigation of the outer surfaces of the building. Additionally, any data that is recorded by the drone can be conveniently stored and later accessed by the user. Ultimately, our goal with this product is to reduce the risk, inconvenience, and potential overhead of having to perform a relatively simple task such as a building inspection.

Building inspections have always been necessary, in almost any industry – nowadays, however, they are almost always performed via the use of a drone. Conventional approaches to building inspection are laborious, costly, and dangerous. Building inspectors may have to traverse obstructive or treacherous terrain, and as a result they may have to use ladders, construction lifts, or even constructed scaffolding. All of these things have associated risks – they can take long amounts of time, incur liabilities which companies have to insure, they can be dangerous for inspectors to use or climb – sometimes there could simply not be enough space to

bring in the necessary equipment. Using an inspection drone, however, inspectors can acquire high-quality videos of buildings without endangering themselves or using any other overhead. This solution offers the added benefit of being able to save the high-quality footage for later review.

Our project brings this idea to the next level. Often the drones that are used are manually controlled, meaning that the operator must constantly be within line of sight and appropriate range of the drone. Our drone, however, is autonomous. Using an integrated flight-controller software, our operator sets the flight path and sends the drone on its way, up and around the building. A gimbal mounted on our drone stabilizes a camera, which records the nearby building and streams the video data back to the ground control station. Within the ground control station, this video data is presented to the operator; meanwhile, our two-stage feature analysis system is run in the background, identifying relevant features, and highlighting them for the user. Identified objects, such as gutters and windows, are outlined in the video feed for the operator to see, and a trained deep neural network identifies whether this specific feature is nominal or defective. If a feature is determined to be nominal, the drone simply continues with its mission. If the feature is defective – that is to say, a window is cracked or a gutter is clogged – the drone pauses its mission and waits for the user to allow it to continue, storing its stabilized video footage.

The driving idea behind this drone is to bring an even more convenient and thorough approach to building inspection, while developing a novel method of machine learning implementation. Our goal is to make an easily-operable, low-risk, and sophisticated solution to the commonplace issue of building inspection.

Design Process

Design Process Summary

Our team designed this system by breaking down responsibilities into development branches, allowing each team member to utilize their individual strengths and specialize in their particular task. We began by individually researching existing methods of drone automation, structural inspection, and machine learning object detection. This led us to some pivotal discoveries, such as the PixHawk 4 flight controller and QGroundControl software, both of which were central to the design of our project. Additionally, it led us toward the use of Darknet's You Only Look Once v4 (YOLOv4) object recognition system and the TensorFlow Python library for our image processing needs. The assembly of the drone was fairly straightforward and was performed fairly early in the development process, after determining which flight controller we intended to use. We bypassed the need to start from scratch and assemble a full parts list by selecting an appropriate drone kit, the HolyBro S500, and making use of the open-source ground-control framework, QGroundControl.

Our original design for the feature detection system was a single-stage object recognition system, You Only Look Once v4 (YOLOv4). However, this alone was unreliable in distinguishing between nominal and defective features, so we decided to implement the second stage of damage severity analysis. This binary classification convolutional neural network stage allowed us to input cropped images containing structural features and determine whether they were nominal or defective. This did not greatly affect the processing power required, but added a level of reliability that could be highly beneficial with more acutely-designed or thoroughly-trained neural networks. We additionally used the model-view-controller code paradigm, which is an architecture commonly used for developing user interfaces. This allowed us to create very modular and adaptable code, which was easy to add onto during the design and testing phase and could be effectively extended for any further applications of this system. Other neural networks for either stage of analysis can be installed in place of the current ones, trained on different datasets or with different parameters, without needing to alter the existing base code or implementation.

Functional Decomposition

Our software design is broken down into three major components: the model, the view, and the controller, as defined by the model-view-controller architecture depicted in **Appendix D**. The model portion of the project houses the entire back-end. This part is responsible for all of the internal processing, in our case the image processing and machine learning implementation, which is never directly accessed or observed by the user. The view portion of the project acts as

the front-end, interacting directly with the user and displaying the actual outputs from the system. The controller, then, moderates the connection between the view and the model. The interactions of these components makes up the most fundamental structure of our system.

1. Model:
    a. Video Stream - The framework for video frames in and out.
    b. YOLO - The image detection neural network.
    c. Classifier - Classifies whether a feature is damaged or not.
    d. Action Generation - Generates a drone action based on a video input.
2. View:
    a. GUI View - The connection between the user interface and the rest of the code.
    b. Window - The graphical user interface layout.
3. Controller:
    a. Controller - An abstraction layer between the view and model. Controls the flow of data from front-end to back-end.

Prototype Findings

Our team implemented two low-level prototypes early in the design phase to model several of our earlier branches of development: image classification and drone navigation/path planning. This was performed so that members of the team could begin researching and specializing into their specific fields of development, which are described below.

1. Classification Network
    a. One goal that we pursued was the development of a neural network that can receive an image or set of images and detect the location and size of specific objects, returning an image with a rectangle drawn around the object. These processes will determine the areas of concern to report back to the operator, as well as prompt the drone to perform a closer inspection on certain regions. During the development of this prototype we attempted to learn several things: firstly, the very basics of machine learning so that we can branch into our specific application of the technology. The goal is to learn how to create a neural network that can identify the location and size of objects within images and draw a box around them. We expected the entire prototype to be challenging, as we had never worked with machine learning before, so it is entirely new. This prototype should provide a strong basis upon which we can continue to develop our machine learning protocol. This is one of the core features of our product, so it is good to begin development early.
    b. Ultimately, we were not able to successfully create a neural network that was able to detect features within images. We got close, to the extent that the neural

network is currently able to identify whether an image contains certain features – we used Garfield the Cat as an experiment, since there are tons of slightly varied pictures of him readily available on the internet where he is the focus of the image. The idea was to extend this implementation in order to be able to identify the size and location of an object (in this case, draw a rectangle around Garfield). This prototype was originally a failure; however, we did learn quite a lot about the implementation of TensorFlow in Python. Ultimately, we did return to this prototype much later though as we decided to implement the second stage of our feature analysis system - the binary classification. We then used some of this research to jumpstart our development in this field.

2. Flight Simulator
    a. To ensure safe and responsible operation, our first priority was researching and configuring a simulation environment. Fortunately, PX4 has out of the box simulation integration with several great options. We chose “Gazebo” simulator because it is open source and supports several real-world enhancements such as GPS noise and configurable scenes. Most importantly, we expected to learn how to send control packets to the drone through the simulation environment. This would be absolutely critical for testing our autonomous control systems.
    b. We had great success while researching the simulation options for our drone. The Gazebo simulator has many great features such as: real world failure assertion, GPS noise, environment factors, real time control and command input, and sensor configuration. The combination of these features will allow us to test all of our autonomous building inspection code without needing to risk our hardware or safety. Ultimately, we were able to compile and simulate the flight control stack and send it commands to take off and land. We were also able to create a small graphical environment to explore in the future. The logical next step in developing this prototype was exploring what programming interfaces are available so that our machine learning model could receive images from the simulation and our autonomous flight control software could send control commands back.

Final Design

System Architecture

The design of our project is heavily reliant on both hardware and software components. The first of the following figures shows our overall hardware architecture, which is primarily separated into two fields: the drone apparatus and the ground control station. The drone is a quadcopter UAV, which is outfitted with a number of sensors and transmitters/receivers. Each of the four motors is controlled by an electronic speed controller. All of the power for the drone and auxiliary equipment is supplied by a lithium-polymer battery, and it is regulated and distributed by a power distribution board. The flight controller mounted on the drone receives positional data from its GPS antenna, accelerometer, gyroscope, and magnetometer, and it transmits this data to the ground control station via its telemetry radio. The drone carries a radio control receiver to be manually controlled by a remote controller. A gimbal system is mounted on the drone, carrying a GoPro Hero 4 camera, which streams analog video signals via radio back to a receiver connected to the ground station. Our fully constructed drone can be seen in **Appendix E.** The ground station consists of a laptop running the QGroundControl flight control software, connected to a telemetry radio for sending mission commands as well as a radio receiver for video data. The machine learning object recognition systems are implemented within the ground control software stack on the ground control station.

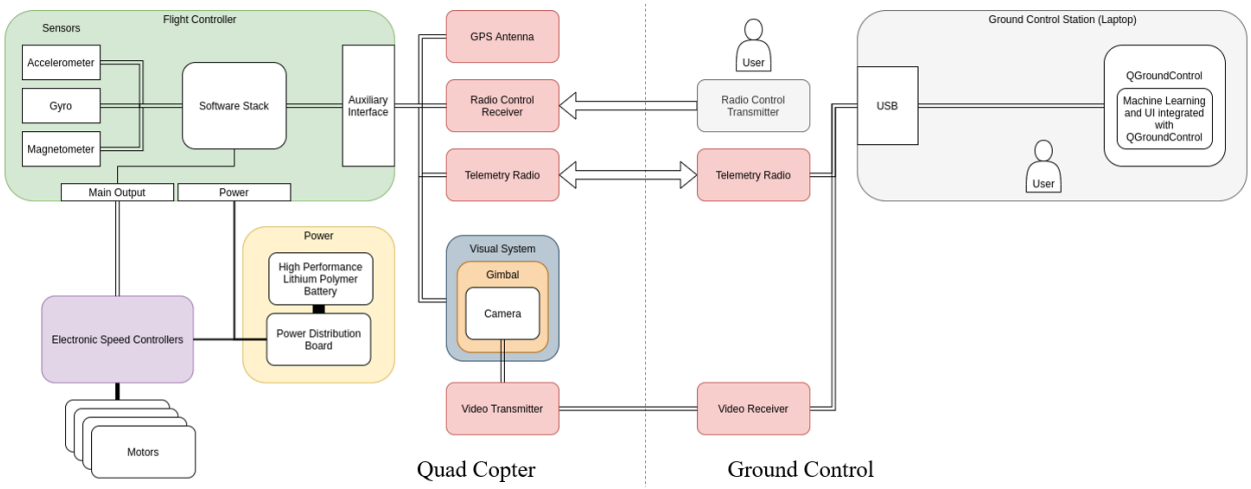


Figure 1. Hardware System Architecture

Secondly, the two primary components of our software are the YOLOv4 object recognition system and the severity classification models, which are implemented using TensorFlow and OpenCV in Python. The overall structure of our software follows the MVC

architecture, which is a paradigm used to implement user interfaces by separating the project into modular components and smoothly controlling and standardizing the flow of data through the system as represented in Appendix C. This data flow will occur within the ground control station laptop, which receives the video data wirelessly from the drone, and can be seen in Appendix A. This video data is sent to our YOLOv4 model, which annotates the video feed and displays it to the user in QGroundControl. At the same time, it outputs snipped images of structural features to the damage classification network using OpenCV. This classification process will determine what inputs are prompted from the user, i.e., whether they want to continue the mission or pause to keep inspecting the feature. These commands will be transmitted to the drone via MAVLink and will be displayed to the user in QGroundControl.

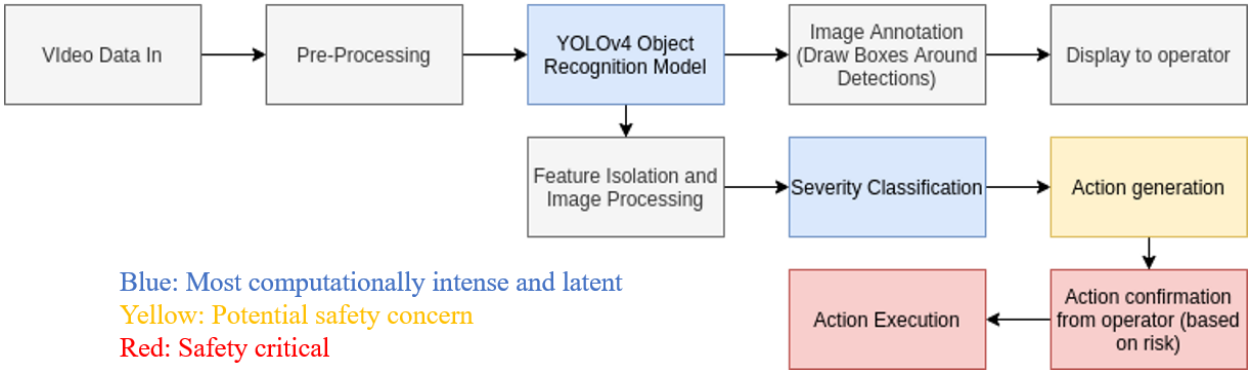


Figure 2. Software Data Flow

Major Components

1. Drone Kit
    a. We used the HolyBro S500 Drone Kit. This kit came equipped with four motors, four electronic speed controllers (ESCs), the drone frame, a power distribution board, a telemetry radio, a GPS antenna, and the PixHawk 4 flight controller. We further installed a remote-control receiver for manual control of the drone, a 3-axis gimbal, a GoPro Hero 4 video camera, and a lithium polymer battery to power the entire drone.
2. Ground Control Software
    a. We used the QGroundControl open-source flight control program. This program pairs with the drone wirelessly and allows the user to configure all of its parameters, designate flight modes, and plan missions. This software also receives all of the data being transmitted from the drone, allowing the user to see the drone's video feed and positional information. This acts as the base-level user interface, around which we designed a settings GUI for run-time configuration.
3. Object Recognition System

   a. We used the You Only Look Once v4 object recognition system, found within the Darknet GitHub repository at https://github.com/AlexeyAB/darknet. We extended this high-level, open-source software and trained it to recognize two classes: windows and gutters. We utilized Google Colaboratory and the Cloud VM GPU acceleration to train our object detection system more quickly and efficiently than our local non-CUDA-enabled processors could. This system outputs the bounding boxes of detected objects, which are then cropped and sent to our damage classifiers.

4. Visual Damage Classifier
   a. We used the TensorFlow and OpenCV python libraries to create binary classification neural networks which distinguish nominal versus faulty features. These classifiers utilize Sobel edge detection and convolutional layers and are trained on a custom dataset. We have two differently-configured neural networks that we use - one for each feature type. The output from this system is the nominal vs. defective classification, which prompts the subsequent actions taken by the drone and user.

Results

Requirements

The following figure shows the list of requirements that we designated for our project, as well as the corresponding type and status of testing. Certain requirements which were more critical to our project are marked with an asterisk next to their requirement number.

Type of Test	Status	Req #	Requirement
		1	**User Interface**
Inspection		1.1*	The drone must be operable using a graphical user interface
		1.1.1	Allows the operator to navigate the drone, configure inspection or navigation settings, or access data from the drone's flight.
		1.1.2	This should be written in a common programming language such as Python or MATLAB
Inspection		1.2	The drone should be operable and configurable wirelessly
Inspection		1.3	The user will be able to see both the drone's positional data and video stream during its flight, and be able to issue commands mid-mission
		2	**Image Processing**
Integration		2.1*	The drone will use image processing in order to isolate and identify features relevant to the operator, which should be at least 80% accurate
Inspection		2.1.1	There will be four recognizable features relevant to our project: gutters, windows, walls, and roofs
		2.2	Each of the structural features will be initially identified by a YOLOv4 neural network, with a reasonable degree of speed and accuracy
Step-by-step		2.2.1*	Each YOLOv4 feature should be recognizable with at least 90% accuracy
		2.2.2	The YOLOv4 system should output appropriately cropped images to be classified for damage severity
Inspection		2.2.3	In order to accomplish this, the team will collect at least 300 images for each category
		2.3	A variety of classification neural networks will be run on the detected features, specialized for each feature
Step-by-step		2.3.1	The damage classifiers should be at least 90% accurate in distinguishing between faulty and intact features
Step-by-step		2.4*	Feature detection within the video feed will trigger appropriate commands to be sent via MAVLink
		2.4.1	The YOLOv4 object detection event will instruct the drone to halt its mission in order to investigate
		2.4.2	If the feature is classified as defective, the drone will prompt the user to give input, otherwise, the mission will continue
		3	**Drone Assembly**
		3.1	LiPo battery capable of powering drone flight controller, motors, and auxiliary devices
		3.2	Maneuverable camera mount, capable of 3 degrees of freedom
		3.3	Flight control systems should strive to communicate over a single unified protocol
Inspection		3.4	Power should be supplied to all auxiliary components via the same power distribution board
Inspection		3.5	Drone will stream analog video data in real time to the ground control station
		4	**Flight Path Navigation**
Inspection		4.1	The drone must be operable using a remote controller
Inspection		4.2	The drone must be capable of semi-autonomous flight (not directly navigated by remote control)
		4.2.1	The mission flight mode will receive and quickly react to events in the form of MAVLink instructions
Inspection		4.2.2	The drone is capable of safe takeoff/landing protocols
		4.2.3	The drone's mission flight mode allows for the designation of regions of interest for surveying

Figure 3. List of System Requirements

Major Test Results

Our team conducted two major tests which verified several of our more critical requirements.

1. YOLOv4 Object Recognition

The first major test that we performed was the YOLOv4 object recognition accuracy test. We trained the YOLO model three times, using varying numbers of classes and sizes of datasets. The process was performed using a Jupyter notebook in Google Colaboratory, which allows users to access a graphics processing unit (GPU) via the use of a virtual machine. These models

each took over 8 hours to train, even with the GPU acceleration, so we generally trained them overnight.
The first model that we trained was only trained to recognize windows. The image dataset was fairly low – at around 150 images – as it was our initial test on learning how to train a neural network. It was fairly accurate, considering its small training dataset, with an average precision (AP) of 73%, since it was only set to recognize windows.
The second model that we trained was trained to recognize windows, gutters, roofs, and walls, and was quite inaccurate. The dataset for the walls and roofs was practically nonexistent, being around 25 images each, and this model took much longer to train due to having 4 classes to attempt to recognize. The windows and gutters had about 200 and 100 images, respectively, in their training sets, and had a mean AP of around 70%, which we believe was somewhat reduced by the attempts to identify other classes.
The third model that we trained was only trained to recognize windows and gutters, since we had the most data for those two sets – 300 and 100, respectively – and they have the most recognizable features. This was significantly quicker to train than the previous, since there were half as many features, and it was more accurate as well. The mean average precision was 78% with this model, with an AP of 72% for gutters and 84% for windows. This model was more reasonably functional and served well as a proof of concept since it could reliably find windows, even with relatively low confidence values.

Major Test 2: Damage Severity Classifier Optimization

Our second major test was a black box matrix unit test designed to help us determine the most effective configuration for our convolutional neural networks (CNNs). CNNs are fundamentally made up of convolutional layers and dense layers, which are implemented with different activation protocols, different numbers of nodes, and varying levels of compression. The only way to effectively design a CNN is by trial and error since it cannot necessarily be calculated based on the image dataset available. As such we divided our image datasets for windows and gutters into defective and nominal categories for training. Using a series of nested for-loops, we configured and trained 27 different models on the same image datasets. We used an image input size of 128x128 pixels, normalizing the training and testing data, as necessary. We used a batch size of 8, due to relatively small datasets of 300 windows and 150 gutters, and 15 epochs to avoid overfitting, as we have found most effective previously. We performed this process on both of our datasets, allowing us to find that the model with 3 convolutional layers, 32 nodes per layer, and 2 dense layers was the most effective for our window dataset with a validation loss of 0.4803 and a validation accuracy of 80.6%, as seen in the table below. Similarly, when run on the gutter dataset, we found that a CNN with 3 convolutional layers, 128 nodes per layer, and 0 dense layers was the most effective with a validation loss of 0.5007 and validation accuracy of 70.73%, for a mean average precision of 75%. This was below the 90%

mAP threshold that we were attempting to achieve; however, once again, it was accurate enough to be functional for the purposes of testing the integration of all of the components together. A table showing the inputs and outputs of this test when implemented on our gutter dataset can be found in **Appendix C.**

Results Analysis

All of the basic functionalities were completely effective under the test conditions. The core of our software design, primarily developed by Daniel, was very modular and organized, which allowed us to efficiently design each file systematically and cleanly. The overall implementation worked perfectly well, as did the general flow of image data since the high-level wrapper code ensures that all of the inputs and outputs are normalized and in the proper format. The video input stream, YOLO output cropping function, feature isolation and image filtering, annotated display, and action generation all work completely nominally. We were able to store and implement the YOLO and TensorFlow models and weights files and retrain them as needed without having to change the wrapper code. In a controlled environment, with high quality images and videos, the system worked fairly well, bounded only by the accuracy and reliability of our neural networks. We were somewhat concerned since the system did not seem to function as well on raw data that we collected using the drone flights – data that was less curated. Ultimately, the YOLOv4 network had a mean average precision of about 79% and the classifier networks had a mean average precision of about 72%, which is only about a 57% overall precision.

The results were relatively close to what we expected. We were disappointed to find that the neural network accuracies were lower than we had originally intended them to be. We had hoped to achieve at least an 80% overall precision, which we were unable to do with such limited datasets. Two of the primary requirements were not met in their entirety – the overall precision and the object recognition precision – since we did not achieve the thresholds that we were aiming for; however, we were able to get close enough to create a functioning system, regardless of accuracy or correctness. The graphical user interface and action generation requirements were met.

Conclusion

Primary Requirements and Results

1. Requirement 1.1 – The drone must be operable using a user interface.
    a. The purpose of the graphical user interface is to configure the drone upon start-up, as well as to ensure a smooth display of information and prompts to the user. This requirement is important to our client because he would like the drone to be theoretically operable by an individual who is not an engineer or software designer. It should be clearly laid-out and easily operable by reading the user manual, which will detail the recommended settings and parameters of the drone. This requirement is a core feature of our project, and if it were not met then this drone would never (theoretically) go into production or distribution, since it would be largely inoperable by the general populace.
2. Requirement 2.1 – The drone will use image processing in order to isolate and identify features relevant to the operator, which should be at least 80% accurate.
    a. The system necessarily will use image processing and feature detection in order to semi-automate the inspection process. In order to ensure that the drone would actually be effective in practice, we have set a baseline accuracy of 80% when both the object recognition and damage classification systems are implemented. This requirement is important to our client because his field of study largely involves image processing and machine learning applications. If the device did not effectively implement machine learning, it would simply be a video-recording drone, which is not particularly innovative or revolutionary. If we were to fall short of the accuracy threshold, the project would not be ruined, as long as we were able to provide proof of concept; however, we would not want to deploy the device in a real-world setting.
3. 2.2.1* Each YOLOv4 feature should be recognizable with at least 90% accuracy.
    a. This requirement indicates that the You Only Look Once v4 (YOLOv4) object recognition system should have a mean average precision (mAP) of 90%. This means that the when the average precisions of all the classes are averaged, the result should be 90%. This is important to our client because it is a central feature of the drone as an inspection and analysis device. If this threshold were not met, once again, the project would not be ruined, but we would not want to deploy the device in a real-world setting.
4. Requirement 2.4 – Feature detection within the video feed will trigger appropriate commands to be sent via MAVLink.
    a. This requirement means that the YOLOv4 object detection event will instruct the drone to halt its mission in order to investigate; if the feature is classified as defective, the drone will prompt the user to give input, otherwise, the mission will

continue. This feature is additionally important to our client in order to distinguish it from a simple video recording drone. The drone should be able to react dynamically to the visual stimuli that it is receiving, otherwise the drone is fairly mundane, and the project is ultimately a failure.

Lessons Learned

Throughout the process of developing our structural inspection drone, we had a lot of things go wrong. During our first unmanned test flight of the drone, an assembly oversight led to the GPS antenna coming loose and falling out of position. As a result, we hit the kill switch to keep the drone from aimlessly flying away. It then fell from the sky and broke. On another occasion, we faced a similar issue where a propeller became unfixed during the flight, resulting in the drone falling to the earth. These setbacks were unfortunate and inconvenient, but we rebuilt and utilized lock tight and other methods to prevent more mechanical failures from happening.

One of the biggest challenges for this project was developing the machine learning models. For all of us, this was our first real time using machine learning for image processing, so it was an area of huge growth for us. We got our models working pretty well, however, without access to vast quantities of good image data, it wasn't possible to get it working as well as it could have. We gathered our own dataset utilizing a mix of images gathered around flagstaff as well as the internet. Finding high quality images of broken or damaged items was very difficult. Many of them were cartoon drawings or otherwise not useful for our set. Additionally, we weren't able to collect many images of broken features around Flagstaff because people usually fix broken things. In the future, the dataset is definitely one of the areas in need of improvement. Doing so would vastly improve the quality of our object detection and image classification, and furthermore the performance of the entire system.

Our tests, for the most part, were well written and strictly defined. We had no real difficulty testing our system according to our tests. One of the particularly frustrating aspects of our tests, however, was that several of them were on our machine learning models. The machine learning tests were so frustrating for two reasons. Firstly, if it isn’t behaving the way you expect, there is no real way to know why. It is simply not possible to look at the model and understand the decisions it makes. As a result, if something is wrong, fixing it is largely guessing and check. Secondly, creating a new model takes a lot of time. Training our object detection network took in the order of days.

Regression testing was done at intervals along the way. We wrote software unit tests to test our software automatically, ensuring that even after a new change was made, the previously tested code still worked. On several occasions, we updated our codebase and introduced new bugs that might not have been otherwise detected. Regression testing is especially important when you are operating a potentially dangerous piece of equipment, such as a quadcopter.

EE486C Capstone Design
Team 2
Skeyes Structural Monitoring Drone
16 April 2021

User Manual

Introduction

Thank you for choosing Team Skeyes for your structural inspection needs - we hope you like the product that we have developed for you! The increasingly commonplace use of drones for the purpose of inspection indicates a strong need for the Skeyes Structural Inspection Drone. We have provided for you here a powerful and easily operated system for performing routine, external maintenance checks on a building, custom-designed to meet your needs. Some of the key highlights include:

- Integrated and accessible user interface
- Self-navigating, autonomous drone flight
- Two-stage feature recognition and analysis system
- Convenient ground-station video stream display

The purpose of this user manual is to help you, the client, successfully use and maintain the Skeyes Structural Inspection Drone product in your business context going forward. Our aim is to make sure that you are able to benefit from and further extend our product for many years to come!

Building inspections have always been necessary, in almost any industry – nowadays, however, they are almost always performed via the use of a drone. Conventional approaches to building inspection are laborious, costly, and dangerous. Building inspectors may have to traverse obstructive or treacherous terrain, and as a result they may have to use ladders, construction lifts, or even constructed scaffolding. All of these things have associated risks – they can take long amounts of time, incur liabilities which companies have to insure, they can be dangerous for inspectors to use or climb – sometimes there may not be enough space to bring in the necessary equipment. Using an inspection drone, however, inspectors can acquire high-quality videos of buildings without endangering themselves or using any other overhead. This solution offers the added benefit of being able to save the high-quality footage for later review.

The Skeyes Structural Inspection Drone brings this idea to the next level. Most drones used within the industry are manually controlled, meaning that the operator must constantly be within line of sight and appropriate range of the drone. Furthermore, the operator must not only know how to fly a drone via remote control but also be skilled at piloting a drone to the extent that they are able to navigate around a building and thoroughly survey the exterior. Hence, the goal of our project is to create a device that incurs no risk upon the operator, while still being effective, easily implemented, affordable and, ultimately, a novel application.

Hence, we have designed our drone to be autonomous. The Skeyes Structural Monitoring Drone is divided into three primary subsystems. The first subsystem is the drone itself. The drone is outfitted with a variety of sensors, which allow it to analyze, record, and transmit all kinds of positional information about itself. Several transmitters and receivers allow it to send data, receive orders, and undergo manual control. The onboard flight-controller moderates the interactions between all of these components and sends instructions to the electronic speed controllers which control the motors. It also controls the gimbal, which holds the video camera used to capture and transmit data back to the ground control station. The second subsystem is the ground control station. The ground control station is implemented on the operator's laptop and allows the user to configure the drone, observe the video feed, provide instructions, and run the processing occurring within. This is operated via the use of a graphical interface, which allows the user to easily configure certain parameters, and the QGroundControl flight control software, which interfaces directly with the flight controller on the drone. The third subsystem is machine learning feature analysis subsystem, which is a software stack implemented within the ground control subsystem. The two-stage feature analysis program uses an object recognition system paired with specialized convolutional neural networks in order to find and evaluate features on buildings under inspection. These were trained using custom datasets for windows and gutters which we compiled using both the internet and our own cameras.

Our team designed this system by breaking down responsibilities into development branches, allowing each team member to utilize their individual strengths and specialize in their particular task. The assembly of the drone was fairly straightforward and was performed fairly early in the development process. We bypassed the need to start from scratch and assemble a full parts list by selecting an appropriate drone kit, the HolyBro S500, and making use of open-source software for our base ground-control framework, QGroundControl. Our original design for the feature detection system was a single-stage object recognition system, You Only Look Once v4 (YOLOv4). However, this alone was unreliable in distinguishing between nominal and defective features, so we decided to implement the second stage of damage severity analysis. This binary classification convolutional neural network stage allowed us to input cropped images containing structural features and determine whether they were nominal or defective. This did not greatly affect the processing power required, but added a level of reliability that could be highly beneficial with more acutely-designed or thoroughly-trained neural networks. We additionally used the model-view-controller code paradigm, which is an architecture commonly used for developing user interfaces. This allowed us to create very modular and adaptable code, which was easy to add onto during the design and testing phase and could be effectively extended for any further applications of this system. Other neural networks for either stage of analysis can be installed in place of the current ones, trained on different datasets or with different parameters, without needing to alter the existing base code or

implementation. This is another feature that we were particularly striving for with our design, since we were somewhat limited in our dataset. The ability for future teams to implement their own network and other plugins with our system is what makes the Skeyes Structural Monitoring Drone a great choice for you!

Installation

This following section will describe the installation of the skeyes structural inspection drone application. All required dependencies will be detailed; however, optional dependencies will not be covered. Please note that the directions are for and this application has only been tested on *nix systems. Other operating systems may exhibit unexpected behaviors.

1. Open a terminal window.
2. Clone the structural-drone repository from github with the command and change directory.
   `git clone https://github.com/djcopley/structural-drone && cd structural-drone`
3. Run the prerequisite installation script. This script will download and compile GStreamer and OpenCV with the GStreamer and Qt backend, and then install.
   `sh ./prereq-install.sh`
4. Install the structural drone python package via the setup.py installation script.
   `python3 setup.py install`
5. Optionally, Install NVIDIA graphics drivers and the CUDA toolkit for hardware acceleration (outside the scope of this document)

Configuration and Use

The following section will describe tuning parameters, configuration, and use of the skeyes structural inspection drone and application.

Before each flight

1. Preflight checklist
   a. Propellers are tightly fastened
   b. All devices are fastened securely (Especially battery and GPS)
   c. Make any hazards are clear of drone
   d. Ensure that drone has GPS signal and connection to remote
   e. Configure failsafes
2. Plug in usb telemetry radio to ground control station
3. Ensure ground control station is connected to GoPro wifi network
4. Run the “Skeyes” graphical application and wait for the user interface to appear.
5. Launch QGroundControl

Skeyes Software Configuration

Main Menu

The main screen contains a list of building features that the neural networks are trained to recognize. From this menu, you can control the action taken by the drone when any of these features are detected damaged.

Settings Menu

Enable / Disable UDP Stream Checkbox

1. UDP IP Address
   This option allows the user to enter an IP address to stream to. This will be the device where the processed video shows up.
2. UDP Port
   This option allows the user to enter a UDP port to stream to. Make sure this matches with the port setting in QGroundControl.

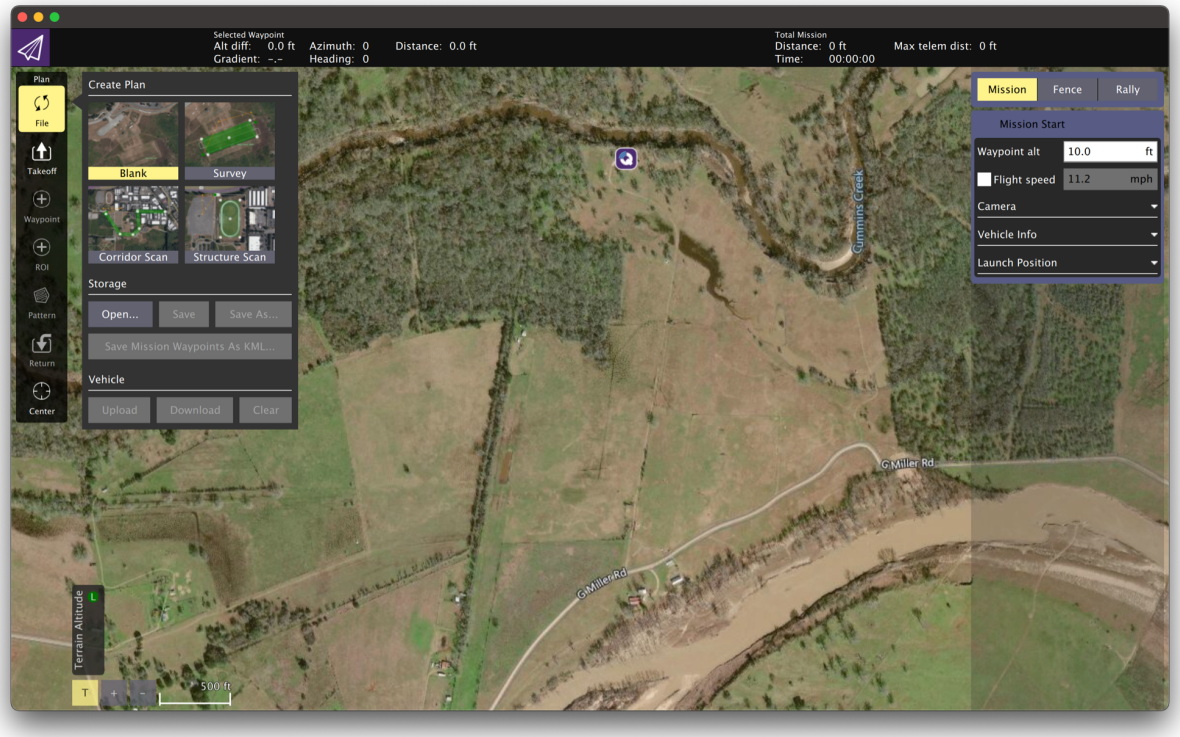
Enable / Disable File Logging

1. File Path
   This option allows the user to specify a file path to save

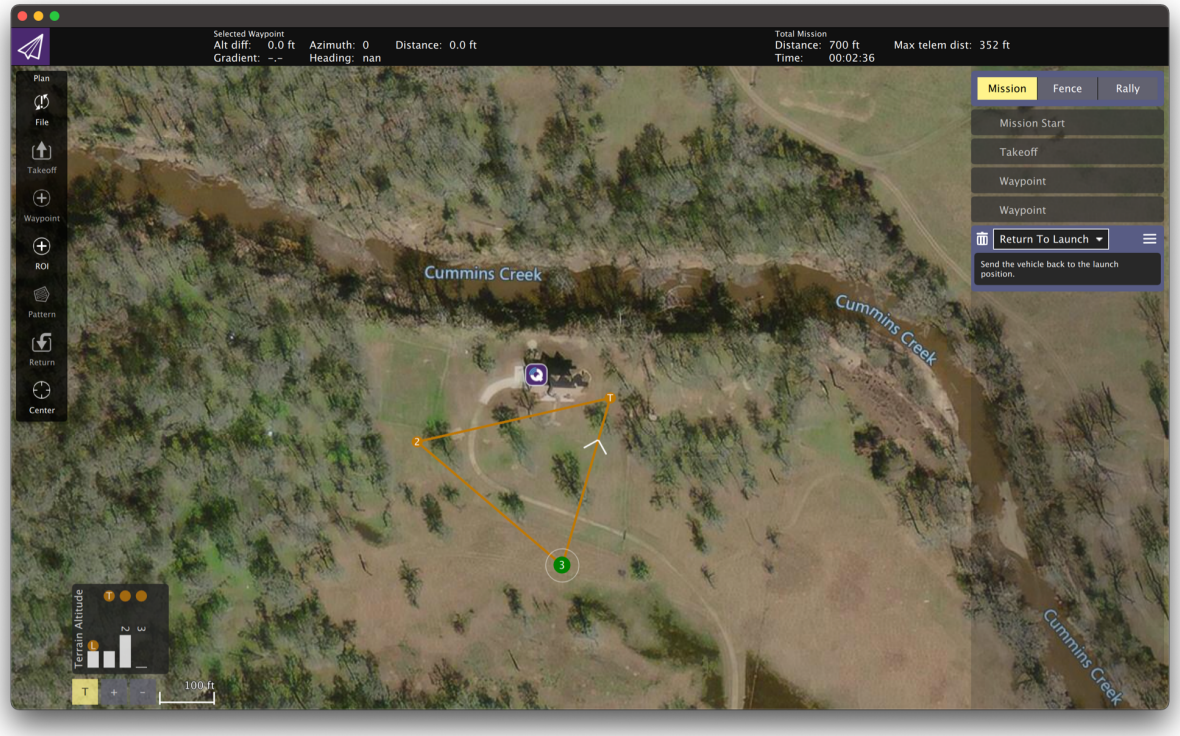
QGroundControl Mission Planning

The basic steps to create a mission are:

1. Change to *Planning View*.



2. Add a takeoff point,waypoints or commands, and a landing point to the mission. Edit as needed.



3. Upload the mission to the vehicle.
4. Change to *Fly View* and begin the mission.

Maintenance

The following section will detail routine maintenance for the skeyes structural inspection drone. Barring a significant mechanical failure, the only real maintenance the user should have to do is charge the battery and switch the propellers. The processes for each are detailed below.

1. Charging battery
    a. Undo battery velcro strap
    b. Unplug battery
    c. Remove battery from quadcopter
    d. Place battery in LIPO battery bag to mitigate unlikely fire event
    e. Connect to LIPO Charger and Charge
2. Replacing propellers (Only needs to happen if propellers have chips or cracks)
    a. Unscrew the damaged propellers

        i. Propellers are counter threaded to the way they turn
    b. Screw the new propellers on

Troubleshooting Operation

Sometimes QGroundControl can interfere with the GoPro video stream. If the skeyes GUI fails to launch, start by quitting both QGroundControl and the Skeyes application. Next make sure that the computer is connected to the GoPro wifi network. Next, launch the skeyes application and wait for the graphical user interface to display. Finally, start QGroundControl.

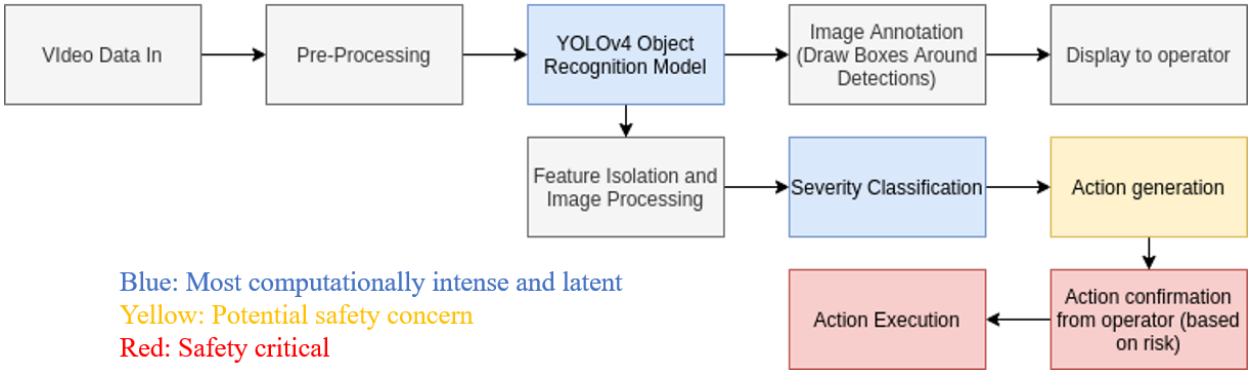
Conclusion

We hope that you are satisfied with the product that we have created for you. Designing, developing, and creating this drone has been a very interesting and informative experience for all every member of Team Skeyes. We wish our client, Dr. Razi, many happy years of use and development with our product, and we hope that future teams are able to take the Skeyes Structural Monitoring Drone to even greater heights. It has been our joy and honor to work on this project, and we thank you for choosing Team Skeyes!

Appendices

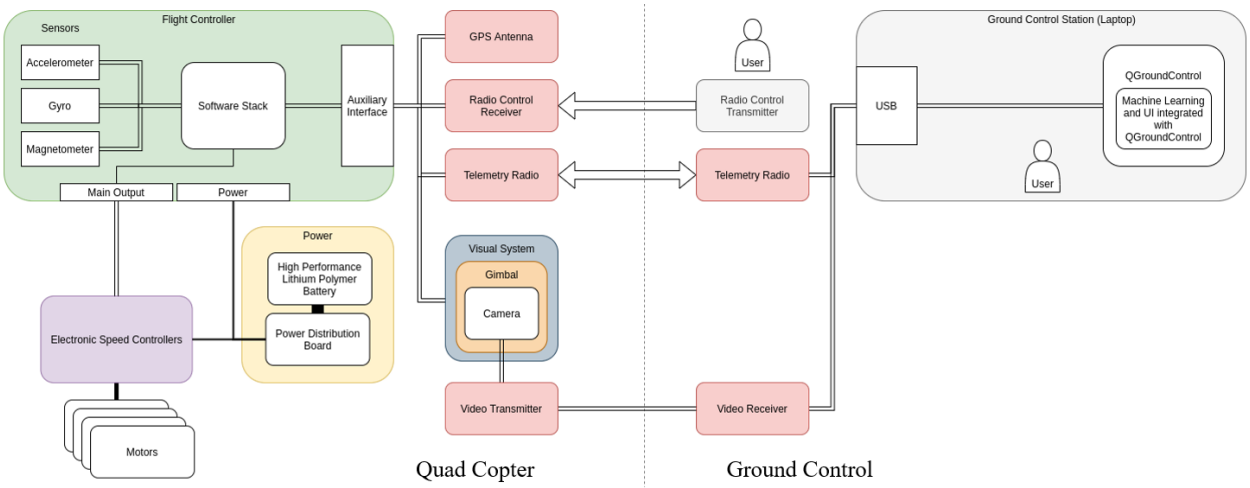
Appendix A – Software System Architecture

This diagram shows the flow of data between software blocks from video input to action output and user display.



Appendix B – Hardware System Architecture

This diagram shows the overall structure of our system, including all of the hardware components and their connections to the ground control station.



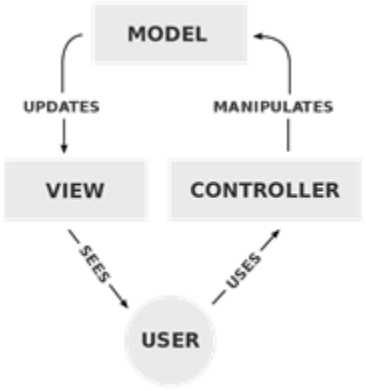
Appendix C - Gutter CNN Configuration Results

The below table shows the inputs and outputs of the program used to test 27 different configurations of convolutional neural networks used to identify nominal versus defective gutters. These configurations were found by using different permutations of one to three convolutional layers; 32, 64, or 128 nodes per layer; and zero to two dense layers.

	Input			Results
Test	Number of Convolutional Layers	Number of Nodes Per Laye	Number of Dense Layers	Validation Loss - Validation Accuracy
1	1	32	0	1.2203 - 0.6511
2	2	32	0	0.6418 - 0.6889
3	3	32	0	0.5105 - 0.7642
4	1	64	0	1.3210 - 0.6410
5	2	64	0	0.6124 - 0.7161
6	3	64	0	0.5116 - 0.7724
7	1	128	0	1.4319 - 0.6415
8	2	128	0	0.6958 - 0.6288
9	3	128	0	0.6866 - 0.7345
10	1	32	1	0.6932 - 0.4973
11	2	32	1	0.6932 - 0.4973
12	3	32	1	0.5162 - 0.7703
13	1	64	1	0.6931 - 0.5029
14	2	64	1	1.2960 - 0.6976
15	3	64	1	0.5247 - 0.7782
16	1	128	1	1.4192 - 0.6581
17	2	128	1	0.7285 - 0.5406
18	3	128	1	0.4830 - 0.8060
19	1	32	2	1.0445 - 0.6843
20	2	32	2	0.5616 - 0.7500
21	3	32	2	0.4792 - 0.7831
22	1	64	2	1.3918 - 0.6716
23	2	64	2	0.7490 - 0.7324
24	3	64	2	0.5257 - 0.7972
25	1	128	2	1.3729 - 0.6811
26	2	128	2	1.2811 - 0.6729
27	3	128	2	0.5495 - 0.7897

Appendix D – Model-View-Controller Program Architecture

This diagram represents the model, view, controller design, which is extremely powerful because both the model (back-end) and view (front-end) are completely oblivious of each other. As a result, changes can be easily made to either one without updating the other.



Appendix E - Assembled Drone

The following images depict our fully-assembled drone, with all of its auxiliary components on-board. The first image is the front view of the drone, featuring the camera and gimbal. The second image is the profile view of the drone, which displays the battery pack and mounting of the gimbal.

Drone Frontal View

Drone Profile View

Appendix F - System Software

The following GitHub repository contains the entirety of our code for this project. The master branch holds the final versions of all of the main code, i.e. not that which was used for testing or the training of neural networks.

GitHub Repository: https://github.com/djcopley/structural-drone