University of Arkansas – CSCE Department
Capstone I – Final Proposal – Spring 2022

Automatic Drone Tracking

Student Team:

Andre Fuentes, Parker Weber, Zachery Gansz, Byron Denham, Corbett Stephens

Champion Advisor:

Prof. Khoa Luu

Abstract

The aim of this project is to bring facial tracking technology to drones. This will be accomplished by the development of a computer vision algorithm for facial detection that will interface with and control the drone. The drone's camera will communicate with a Haar cascade object detection model to gather information about the presence and location of a human face and make decisions about how the drone should be maneuvered to locate and track the face. The development of the technology in this project will have benefits for surveillance, tracking, and the film industry.

1.0 Problem

The integration of facial detection and tracking technology with drones will help give further practical use to existing ideas and technologies in the area of artificial intelligence. This combining of the two areas will help to automate some of the uses drones already have. One example of this is for surveillance. Because of their mobility, drones are already useful for surveillance, but the integration of facial tracking will improve the ability to locate and track criminals. Drones used for this type of tracking would no longer need to be controlled manually and could instead operate automatically. Another area it would be useful for is the film industry. This technology can automate the tracking of actors and actresses in shots taken with drones. One final benefit would be for presenting in a classroom setting. Drones with this facial tracking ability would be able to follow a presenter as they move around during a presentation.

2.0 Objective

The objective of this project is to use computer vision technology to give a drone the ability to recognize and track a human face. The drone will be able to maintain a defined distance from the target's face in order to ensure proper tracking. The drone will accomplish this by using the camera included on it. The drone will respond to and use the Tello API in order to maneuver itself. The developed facial recognition software will work alongside an expert system to interpret the still frames of the drone camera's video and issue responses. The final deliverables will include the facial recognition software, the drone control system (the expert system), and the drone with the attached camera.

3.0 Background

3.1 Key Concepts

A drone is an aircraft without any human crew on board. The drone that is used in this project is a small DJI Tello drone. This drone is a Consumer Off the Shelf (COTS) drone. There is another COTS drone that is used as a backup in case of any damages or hardware malfunctions in the DJI Tello drone. The backup drone is a larger Parrot drone. The decision to move forward with the DJI Tello rather than the Parrot stemmed from two main reasons. The first reason is that the DJI Tello drone has newer hardware, software and it is in better condition. The second reason is that the DJI Tello is smaller with less powerful propellers. Powerful propellers can cause serious harm to objects and people if used improperly. The small body size and propellers of the DJI Tello makes flight testing much safer and easier in a small lab setting. These drones are powered by rechargeable lithium polymer batteries and are made for producing aerial footage. The standard drone is controlled via a radio controller that gives the pilot the ability to remotely maneuver the aircraft within a specified range. These drones come stock with a camera. The camera is used as input data for an algorithm to detect a human's face. The algorithm that is used to detect a human's face is a Haar cascade. The subsequent section will go into more detail about Haar cascades. The DJI Tello drone does not have adequate hardware to run the computation necessary to identify faces on its own. Thus, the drone will need to transmit its camera feed to a computer that is more capable. The algorithm will be able to identify a human face within a video feed processing 20 frames per second. A high frame per second will reduce the latency between the human face's movement and the drone's movement. The DJI Tello drone has an API available that will allow us to view the camera's image and send flight commands remotely via python code. An expert system written in Python will guide the drones flight and keep the human face in center frame. Putting all of these pieces together, the drone will have the capabilities to detect a human face and maneuver itself autonomously in order to keep the face in center frame.

3.2 Related Work

Detectors are a widely discussed topic in machine learning. To be more specific, the detectors being referred to are object detectors that use Haar cascade classifiers. Haar cascade classifiers are a machine learning based approach where a cascade function is trained from a large amount of positive and negative images. The positive and negative images consist of images with faces and without faces respectively. Haar features are used to determine features of an object. An object like the face is assigned Haar features that represent the most relevant features of the face. The Haar features are composed of white and black zones. The arrangement of the white and black zones can determine edge features, line features, and four-rectangle features. For an image, a window of pixels is examined in which a feature is applied, and a single value is obtained by subtracting the sum of the pixels under the white zone from the sum of pixels under the black zone. The integral image method is used to provide accuracy across multiple iterations without the cost of efficiency. The integral method works by using a singular value that represents the sum of a large region. The closer the value is to one, the more likely it is that a Haar feature has been detected. To speed up the process Adaboost (adaptive boosting) is used to select the best features among the calculated features. Adaboost uses several weak classifiers to create one strong classifier. It works by adjusting weights of correctly classified and misclassified items until the error is of satisfactory degree. Once a certain degree of error is achieved, a strong classifier can be made. The concept of Cascade of Classifiers can be used to make the process even more efficient. The features are grouped into different stages of classifiers where they are applied one-by-one rather than applying every single feature to a window every time [1]. This improves the efficiency specifically if a window fails in the first stage. In this case, the remaining stages are not tested. If a window passes each stage, then it is considered an object region such as a face region. The Haar cascade classifier, trained on detecting faces, gives a high efficiency method for accurately detecting faces in a video.

Detecting faces from video is a hard task, and there are many variables that come into play in non-lab settings. In lab settings, you can achieve very consistent and optimal lighting of the individual's face. This helps the model detect the face, but in the real world things become much more challenging. Some challenges that can arise are low light settings, over exposure, different skin colors, poor camera quality, complex backgrounds. We may also have issues with the distance from the camera and the orientation [2]. To overcome these issues the system will utilize a high quality camera with automated focus and exposure settings to extract the most accurate information from the scene. The Haar cascade algorithm will be trained on the Flickr-Faces-HQ dataset because it has more variation in terms of age, ethnicity, quality, lighting and background than any other dataset currently available. The variation will improve the drones ability to generalize to many more faces and settings.

The famous company DJI, based in China, has had object tracking in their drones since 2016. They produce the best COTS drone systems that autonomously track objects in motion. This system functions extraordinarily well, but it does come at some cost. The Phantom 4 Pro V2.0, for example, costs around $2,000 for the base model. The premium price tag is due to the thorough research and development put into the software along with the extra sensors of the drone. The DJI drone uses GPS and a forward and downward vision system that uses many sensors to achieve this task [3]. Our implementation will be a low cost tracking solution that will utilize one camera and a flight controller to achieve facial tracking.

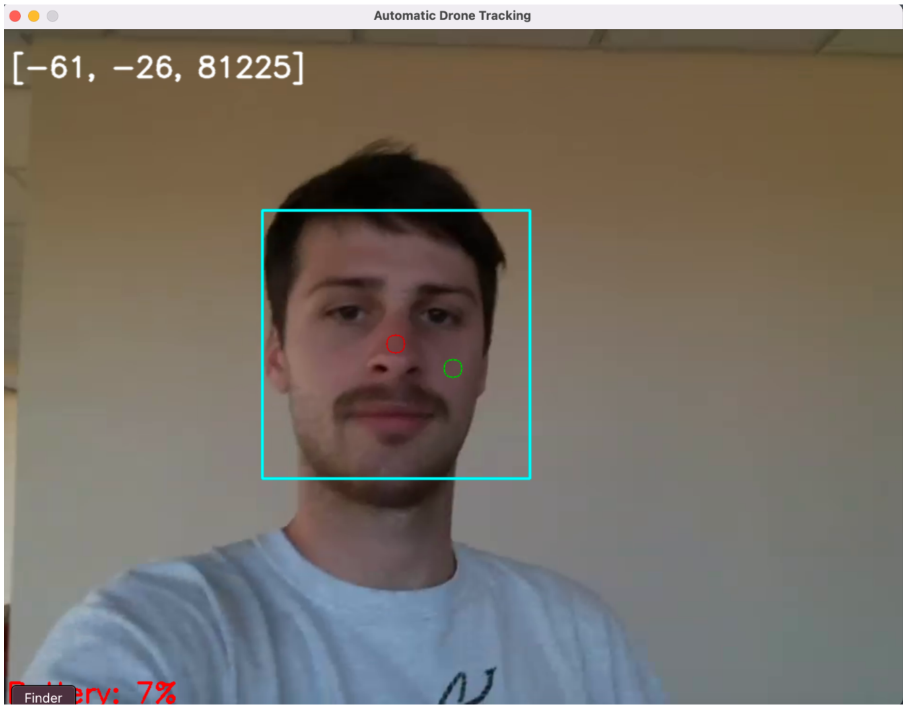
4.0 Design

4.1 Requirements and/or Use Cases and/or Design Goals

- High Quality Camera - to provide a high resolution image, as well as function in low light scenarios and be able to clearly depict a person's face from a distance of 5 to 10 feet away.
- Drone Control System - the backend must be able to properly interpret data given from the facial recognition software in order to have the correct move commands given.
- Drone Stability - the drone must be able to provide smooth and consistent movement.
- Facial Recognition Software - the system must be able to correctly identify human faces of all races and overcome some facial obstructions, such as glasses.
- Environment adaptability - the drone must be able to simultaneously track a human face while also avoiding environmental collision/disruption.
- Processing Power - enough computing power to issue at least 20 commands per second, as the system will only be receiving about 20 frames per second of information.

4.2 High Level Architecture

The system will contain three main parts - the drone, an expert system, and an artificial intelligence backend for facial detection. The three will communicate such that the drone sends real-time information to the backend, and the backend interprets this real-time information, processes it, and then the expert system instructs the drone on its next actions. Together, these three pieces will enable the creation of a system that has capabilities beyond normal, on-board drone systems.



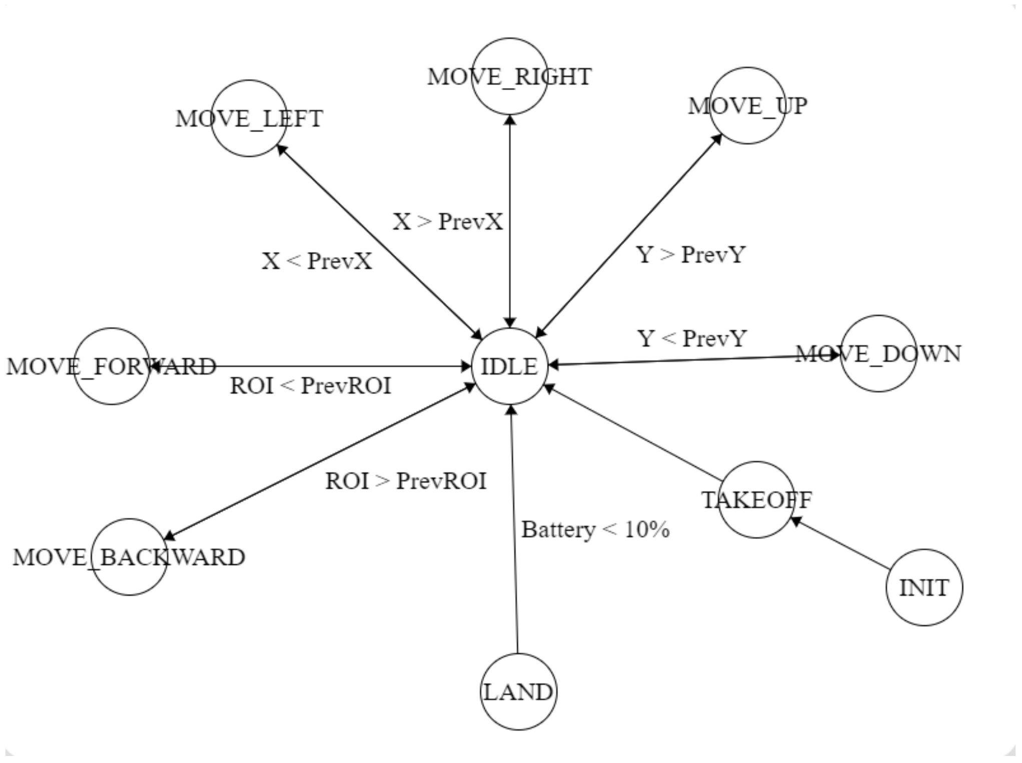
The artificial intelligence backend utilizes the "OpenCV" Python module, a library designed to aid in real-time computer vision, to create a trainable classifier for facial detection and draw pertinent information regarding the facial detection to the video stream received from the drone. The classifier used to detect faces in the drone's video will remain static throughout its execution and not include online learning due to limited hardware resources. The classifier, "face_cascade", is created using OpenCV's CascadeClassifier function and utilizes pretrained weights in the "haarcascade_frontalface_defualt.xml" file. During the program's execution, the center of the screen's coordinates are calculated and a circle is drawn at the center of the screen using OpenCV's "circle" method. In order to apply the classifier to the current video frame from the loop, the frame image must be converted to greyscale. This was accomplished using OpenCV's "cvtColor" method, which converts an image to a new, specified color space. Next, the facial detection was performed on the frame using OpenCV's "detectMultiScale" on the classifier and passing the frame to it. This method returns a list of four-tuples containing the 'x' and 'y' coordinates of the face, as well as its width and height. This list is passed to a function that edits the list to only contain the face with the largest ROI, allowing the drone to target the closest face for tracking. The values of the targeted face's tuple were used in conjunction with OpenCV's "circle" and "rectangle" methods to draw a rectangle around the face, as well as a circle at the center of the face. The width and height, 'w' and 'h', are used to calculate the area of the rectangle which will be used as the region of interest (ROI) value. Lastly, the coordinates of the detected face, as well as the ROI value, are placed in the top left corner of the frame using OpenCV's "putText" method. The coordinates of the face and the ROI value are what is passed to the expert system in order for it to make decisions regarding the drone’s movements.

There are multiple improvements that can be made to the artificial intelligence backend. Currently, the pretrained weights are designed for frontal facial recognition and the program has difficulty continuing to recognize a face as it turns away from the drone’s camera. To enhance the functionality of the face recognition, it will be necessary to find pretrained weights for recognizing faces at multiple angles and integrate the ability to recognize faces at angles such that a targeted face that turns away will not go undetected by the drone for an extended period before the face at an angle is recognized. Additionally, the classifier may toggle between multiple faces if there is more than one face present in the drone’s camera. Ideally, the face recognition would choose one of the faces visible from the camera and fixate on maintaining recognition on that face for tracking. Lastly, the current model functions best in settings with an ample amount of light exposure on the face, such that turning away from a light source has the potential to break the detection of the model on a face. It may also be useful to find pretrained weights for face detection in different light settings.

The controller implementation is what is considered the driver portion of the code. It utilizes the “djitellopy” repository which can be found on GitHub, a software development website. The djitellopy repository makes integrating the manual control of the drone a lot easier. This repository has all the socket programming functions necessary for issuing commands to the Tello drone and video streaming capabilities with “PyGame”, a set of Python modules intended for the development of multimedia applications. Upon initialization of the controller implementation, a PyGame instance is created for drawing the video frame. A Tello object is also created which sets the initial velocities for the drone. When the program’s “run” method is called, a connection to the drone is established alongside the start of the video stream. The program’s main “while” loop initially listens for “PyGame USEREVENTs”, which would be a keyboard input or the expiration of the PyGame timer. Upon the expiration of the PyGame timer, the update method is called and sends the current values of the different velocity variables for the drone to update itself. Returning to the Inside of the main “while” loop, the face detection is accomplished and updates the current frame in the PyGame window. This process continues until the escape key is pressed. The first iteration of the controller did not use the djitellopy repository, but instead used a different repository that caused a lot of issues. This repository (“Tello-Face-Recognition”) had many issues within the socket programming portion. Commands could not be issued correctly, causing the program to crash. Since the program would crash, the computer that was being used would stay bound to the UDP ports on Tello’s Wi-Fi network. This meant that the next time the program was executed, it would not function, because those ports were already in use. Another reason that the implementation pivoted from the Tello-Face-Recognition repository was that there were problems with keyboard input in the OpenCV library. To attempt to debug this issue, a test program was made to receive video frames from the drone and construct a window with OpenCV. If the user pressed the “q” key, then the program would terminate. This program would not terminate properly either. After trying different test cases, the next idea was to try different versions of OpenCV. This led to testing different versions of OpenCV and even the use of a different package installer. Ultimately, this is how the implementation pivoted to djitellopy. An initial test with a manual control example in the djitellopy repository proved that the use of PyGame allowed for a more fluent video stream and keyboard input. The implementation with the djitellopy repository also allowed for easy integration with the face detection code. The next stage in implementation was to integrate the expert system with the controller implementation to accomplish automatic face tracking. One problem that was faced was that there were too many commands being sent at once. To accommodate for this, a manual delay was added along with a flag that was set whenever a command was sent.

The expert system portion of this design will contain two major components. The first component consists of a Python language finite-state machine. This finite-state machine takes in several outputs from the Artificial Intelligence portion of the system and helps to make decisions based on the face's position relative to the camera. These outputs from the AI backend previously mentioned in this section include the face's 'x' and 'y' coordinates relative to the camera, and the total area of the face, or region of interest (ROI). The finite-state machine then uses these parameters to make decisions on which command to send back to the drone control system. The second major component to this side of the system includes the use of Cython for optimizations. Due to the nature of Python and its lacking ability to create sufficient real-time finite-state machine models, it is beneficial to include the use of Cython. Cython is an optimizing static compiler that has two major uses. The first main use is to act as a bridge between Python and C files. This allows for Python to gain a massive speedup by compiling and calling C code for tasks that rely more on real-time efficiency. A second use for Cython is to convert portions of existing Python code, when possible, into C code. This allows for speedups without having to write an original program in C, while still being able to garner the benefits of the C language's efficiency.

The first component mentioned, the finite-state machine, is implemented in Python. The attempt to implement a model like a finite-state machine into Python introduced several considerations. Typically, finite-state machines are modeled in hardware languages like C, due to the built-in functionality of "switch" statements. These statements make it incredibly efficient and have great ease-of-use. In contrast to this, Python does not have switch case statements. Thus, this requires the implementation that will be described below.



A Python dictionary was declared that contained the name for each state, associating each of them with a numerical value. In this case, because there were ten states, the values ‘1’ to ‘10’ were used: for example, ‘1’ refers to the state declared as “INIT”, and ‘10’ refers to the state declared as “MOVE_BACKWARD”. This dictionary is called within a tick function, ‘FMS_TICK()’, using a ‘get()’ call that relates these states to obtain their own discretely defined functions. For example, “MOVE_BACKWARD”, when called using the “get()” function, then calls the “MOVE_BACKWARD()” state. This then executes instructions to return a corresponding move command variable, “command”, and additionally updates the next state through a global variable, “next_state”. By default, some states always return back to the state defined as “IDLE”. These states include: “MOVE_UP”, “MOVE_DOWN”, “MOVE_LEFT”, “MOVE_RIGHT”, “MOVE_FORWARD”, and “MOVE_BACKWARD”. In the “IDLE” state there are several conditional statements that utilize the values mentioned previously: ‘X’, ‘Y’, and ‘ROI’. These values on their own are not significant, so they are then compared to the cameras center coordinates. For example, if ‘X’ is greater than zero (with zero being the center), then a right move command is sent. Additionally, when comparing ROI values, a minimum threshold is used to have the drone keep at roughly the same distance from the face at all times.

The second major component of the expert system is the Cython code that optimizes the previously described Python finite-state machine. The chosen method to utilize Cython is the conversion of Python code to C code, rather than the bridging of an additional C file into the existing AI Python implementation. This decision was made because bridging a C code implementation using Cython was too complex given the time constraints of the project. To utilize Cython with this method, the Python finite-state machine is converted into a ".pyx" file. Then, the "cythonize" command is imported from the "Cython.Build" library and called on the newly created ".pyx" file containing the finite-state machine code. This converts the functionality of the finite-state machine into identically functioning C code. Upon a call to the original expert system, Cython redirects the computations to the faster implementation it generates in C. This change allows for commands to be issued to the drone controller implementation at a much lower latency. This project's first iteration omits Cython due to interference with the Tello drone's existing API. The current implementation of the provided API does not allow for it to take advantage of the speedup Cython permits. However, the final implementation is designed to be easily transferable to newer, more modifiable drones; thus, Cython is a compatible and useful addition upon expansion of the project.

Further development of the expert system may consist of additional states to account for more possible scenarios that may occur. This includes but is not limited to: a state to better search for the face in the event the face manages to escape the drone camera's frame, a state to adjust the angle of the drone's camera by turning the drone clockwise or counter-clockwise, and a state that lands the drone in its original takeoff location. Additionally, adding functionality for the expert system to make more informed or accurate decisions by assigning a weight to all movement possibilities would allow it to avoid undesirable movements.

The results of this design are reliable communication between the drone and the backend, an optimized and efficient expert system for decision making, and real-time facial detection. The finished product is able to instruct a drone to properly track a human subject in order to maintain a line of sight on their face.

4.3 Risks

Risk	**Risk Reduction**
Injury by drone	Limit how close the drone can get to someone
Radio Frequency Interference	Add Radio frequency filters/relocate equipment

4.4 Tasks

Tasks	Personnel	In Progress	Completed
Finish final proposal	Everyone	Y	Y
Read Haar cascade documentation	Andre, Byron, Corbett	Y	Y
Read Tello drone documentation	Parker, Zachery	Y	Y
Create team website and add personal website links	Corbett	Y	Y
Create personal websites	Everyone	Y	Y
Create GitHub repository for the team	Andre	Y	Y
Develop code to allow the UDP video stream connection to be reusable	Parker	Pivot	Pivot
Develop code to allow the UDP drone control connection to be reusable	Zachery	Pivot	Pivot
Develop code to find height of drone	Corbett	Y	Y
Develop code to find distance from drone camera to face	Byron	Pivot	Pivot
Develop code to allow drone to receive commands from computer input with cv2	Corbett, Andre	Pivot	Pivot
Develop code to allow computer to receive video stream from drone	Corbett, Byron, Andre	Y	Y
Port FSM code to Python	Parker, Zachery	Y	Y
Add FSM code and diagram to GitHub	Parker, Zachery	Y	Y
Complete FSM code	Parker, Zachery	Y	Y
Modify code to use PyGame to issue commands to drone	Corbett, Andre	Y	Y
Integrate face detection and annotations	Byron	Y	Y
Integrate FSM Python code with main code	Parker, Zachery	Y	
Modify code to use PyGame to create video stream window	Corbett, Andre	Y	Y
Prepare preliminary proposal report and slides	Everyone	Y	Y
Integrate face detection and controller code	Corbett, Byron	Y	Y
Modify code to timely issue FSM response to drone	Everyone	Y	Y
Modify code to track one face	Byron	Y	Y
Debug the system	Everyone	Y	Y
Create presentation for the project	Everyone	Y	Y
Present the project	Everyone	Y	Y

4.5 Schedule

Tasks	Dates	In Progress	Completed
Finish final proposal	11/29	Y	Y
Read Haar cascade documentation	11/30-12/6	Y	Y
Read Tello drone documentation	11/30-12/6	Y	Y
Winter Break	12/17-1/18	Y	Y
Create team website and add personal website links	1/18-1/24	Y	Y
Create personal websites	1/18-1/24	Y	Y
Create GitHub repository for the team	1/27-1/31	Y	Y
Develop code to allow the UDP video stream connection to be reusable	1/27-2/10	Pivot	Pivot
Develop code to allow the UDP drone control connection to be reusable	1/27-2/10	Pivot	Pivot
Develop code to find height of drone	1/27-2/10	Y	Y
Develop code to find distance from drone camera to face	1/27-2/10	Pivot	Pivot
Develop code to allow drone to receive commands from computer input with cv2	2/10-2/17	Pivot	Pivot
Develop code to allow computer to receive video stream from drone	2/10-2/21	Y	Y
Port FSM code to Python	2/10-2/23	Y	Y
Add FSM code and diagram to GitHub	2/10-2/15	Y	Y
Modify code to use PyGame to issue commands to drone	2/24-3/3	Y	Y
Integrate face detection and annotations	2/24-3/3	Y	Y
Integrate FSM Python code with main code	2/24-3/10	Y	
Modify code to use PyGame to create video stream window	3/3-3/10	Y	Y
Prepare preliminary proposal report and slides	3/7-3/14	Y	Y
Integrate face detection and controller code	3/14-3/29	Y	Y
Modify code to timely issue FSM response to drone	4/5-4/19	Y	Y
Modify code to track one face	4/23	Y	Y
Debug the system	4/19-4/25	Y	Y
Create presentation for the project	4/25	Y	Y
Present the project	4/26	Y	Y

4.6 Deliverables

- Design Document: A description of each component, including the code for facial recognition, the drones API calls, and the drone.
- Facial Recognition Software: The Python program used to take still-image data from a real-time video and identify human faces, informing the user of region of interest (ROI) coordinates and ROI area relative to the camera.
- Drone Control Software (Expert System): The code that contains the Tello API calls in order to correctly direct the drones movement based on incoming data.
- Drone and camera - The drone and attached camera that were used to provide input information for the Facial Recognition Software.
- Final Report - The final report for the project that includes detailed design architecture, implementation, and technologies used to complete the project.

5.0 Key Personnel

Andre Fuentes – Fuentes is a senior Computer Science major in the Computer Science and Computer Engineering Department at the University of Arkansas. He has completed Artificial Intelligence, Data Mining, Algorithms and Software Engineering. He also works in the Computer Vision Lab under Dr. Ngan Le as a research assistant. Fuentes will be responsible for contributing to the development of the AI model that detects human faces.

Parker Weber – Weber is a senior Computer Engineering and German major in both the College of Engineering and Fullbright College at the University of Arkansas. He has completed Software Engineering, Computer Organization, System Synthesis and Modeling, and Operating Systems. He has worked with full-stack development in order to control full simulation runs of prototype hardware systems. Weber will be responsible for contributing to the development of the drone's decision making system.

Zachery Gansz - Gansz is a senior Computer Engineering major in the Computer Science and Computer Engineering Department at the University of Arkansas. He has completed Computer Organization, System Synthesis and Modeling, Operating Systems, Software Engineering, Embedded Systems, Artificial Intelligence, and GPU Programming. Gansz will be responsible for contributing to the development of the drone's decision making system.

Corbett Stephens – Stephens is a senior Computer Science major in the Computer Science and Computer Engineering Department at the University of Arkansas. He has completed Algorithms, Cloud Computing and Security, Computer Networks, and Software Engineering. Stephens will be responsible for contributing to the development of the AI model that detects human faces and calculates adjustments to update the drone's positioning.

Byron Denham - Denham is a senior Computer Science major in the Computer Science and Computer Engineering Department at the University of Arkansas. He has completed Artificial Intelligence, Data Mining, Software Engineering, and Algorithms. He has also been part of the summer Application Development internship at J.B. Hunt. Denham will be responsible for contributing to the development of the AI model that detects human faces.

Dr. Khoa Luu (Champion) - is currently an Assistant Professor and the Director of Computer Vision and Image Understanding (CVIU) Lab in the Department of Computer Science and Computer Engineering at University of Arkansas, Fayetteville. He is an Associate Editor of IEEE Access Journal. He is teaching Computer Vision, Image Processing and Introduction to Artificial Intelligence courses in CSCE Department at University of Arkansas, Fayetteville. His research interests focus on various topics, including Biometrics, Face Recognition, Tracking, Human Behavior Understanding, Image and Video Processing, Deep Learning and Quantum Machine Learning. He has received four patents and two best paper awards and co-authored 100+ papers in conferences, technical reports, and journals.

6.0 Facilities and Equipment

This project will require a DJI Tello drone, a lab space for flying the drone, and a computer for software development. An available drone already exists for use during the project. The lab space requirement is fulfilled by use of the Computer Vision and Image Understanding (CVIU) lab in JBHT room 447. There are computers in the lab that can be used for software development or personal computers can be used. At the moment, there is not any software that needs to be purchased.

7.0 References

[1] Schapire, Robert E. "*Explaining AdaBoost"*. Princeton University, https://www.cs.princeton.edu/~schapire/papers/explaining-adaboost.pdf

[2] Kumar, Ashu & Kaur, Amandeep & Kumar, Munish. (2019). Face Detection Techniques: A Review. Artificial Intelligence Review. 52. 10.1007/s10462-018-9650-2

[3] DJI Mavic Pro Specs https://www.dji.com/mavic/info

[4] "Cascade Classifier." *OpenCV*, https://docs.opencv.org/3.4/db/d28/tutorial_cascade_classifier.html