B9: Hawkeye

Authors: Alvin Shek, Siddesh Nageswaran, Vedant Parekh
Electrical and Computer Engineering, Carnegie Mellon University

***Abstract*—A drone capable of tracking a target in real time as they move across the ground as well as streaming a live video feed of them to a wearable device.**

***Index Terms*–computer vision, flight control, reconnaissance, state estimation, tracking**

1 INTRODUCTION

Over the last few years, drones have become increasingly popular for both recreational and professional videography. Vloggers can take breathtaking aerial shots of their escapades through mountains and fjords. And during search-and-rescue missions, rescuers can use drone footage to identify victims and monitor their own surroundings to call for backup if needed. Yet, despite the numerous use-cases, one bottleneck stands in the way when it comes to capturing drone footage: manual control. Maneuvering a drone takes skill and concentration. For vloggers, this can be an inconvenience while for mission critical operations, this can prove to be a fatal distraction. Moreover, any task involving user control always has the potential for user error. Thus, the goal of Hawkeye is to be an autonomous drone videographer that prioritizes video quality and tracking accuracy. Since users are putting full faith into Hawkeye's tracking algorithm, the most critical requirements are to update the user position frequently and ensure that the user is centered to the frame at all times. Hawkeye will:

- Capture video of a target and sample it at 15 fps to identify the target's position.
- Use these positions for motion planning to follow the target. The target should always remain within a central bounding box $\frac{1}{3}$ the width and height of the frame.
- Stream 15 fps, 480p+ video of the target to a wearable display.

This takes all the hassle out of the user's hands and enables them to focus on the task at hand. Viewing the live footage and seeing their surroundings is as simple as the flick of a wrist.

There are other drones that have recently arrived in the market which also aim to provide autonomous tracking of a target, such as the Skydio 2 and DJI Mavic Air 2 [1]. However, a few of the shortcomings of these drones is that they are expensive and rely on a smartphone app. The live video feed gets streamed to the user's smartphone, and the user has to navigate through the app to control the drone's tracking. This goes against the philosophy of hands-free convenience by forcing the user to carry a smartphone with them at all times, constantly pull it out to watch the live feed and have to deal with a small screen. Our wearable display and simple button interface aims to provide maximum convenience and minimal interference to the user.

2 DESIGN REQUIREMENTS

Target Detection uses computer vision to predict the center pixel location of the target a given image. Our entire target tracking and motion planning stacks depend on this initial detection, so we need a requirement to guarantee its accuracy. We will measure performance as the standard deviation **in distance** σ_{d1} between true and predicted position, where true position will be hand-labeled by us. We will test using 5 different colored shirts in various lighting conditions to determine the best color to use.
During the actual test, the target will wear a specific shirt and move around at various speeds as a camera records video for one minute. The drone has no involvement in this, allowing us to test computer vision independently. After collecting the data, we will hand-label the target's position in every video frame, run our target detection code, and measure overall standard deviation. Our requirement is the following:

$$2 * \sigma_{d1} \leq \frac{1}{5} W$$

where W denotes camera width. If we divide the image into a 5 x 5 grid, we expect the target to lie in one of these rectangles when the drone is flying at 20 feet high. Thus, we expect the predicted target location to lie within one of these squares centered around the target with 95% confidence.

Figure 1: Example target with 5x5 grid overlay. The red bounding box centered around the target has a length of $\frac{1}{5}$ of the camera width.

Drone Stability and Target Tracking involves both target detection and the drone's motion planning. Performance will be measured as standard deviation in distance σ_{d2} between the target's position and the center of the image, averaged across 30 frames. Intuitively, the drone should try to keep the video footage centered on target as the drone follows the moving target. Since this is testing the drone's ability to keep up with the target, the target will move differently in four tests: stationary, straight forwards, sideways, and then randomly.
Since this test involves motion planning, we will first test in the Gazebo drone simulator by having the drone follow a colored cube in bright lighting conditions. We do not want to crash the drone with newly written code.
At the very end, we will test the full system in the real world as a human target moves around. To evaluate performance, we again will hand-label the target's position on each image frame, split the video into groups of 30 frames, and measure standard deviation across each group.
Our requirement is defined as follows:

$$2 * \sigma_{d2} \leq \frac{1}{3}W$$

where W is width of the image. We define success if the target can lie in the "center region" of the image 95% of the time (hence two standard deviations), which can be visualized as the center of a 3x3 grid on the image:

Figure 2: Example target with 3x3 grid overlay. The drone should always try to keep the target within the center rectangle.

Quality of Captured Video will be 720p, giving a (1280 x 720) image. This is a use-case requirement, where the user should have high quality video saved.

Communication Bandwidth will have hard limits since we have offloaded our computation to a ground computer. Our Pixhawk flight controller expects control commands to stream in at a minimum of 2Hz, so our combined computation and communication latency must not exceed 0.5 seconds for a given timestep. Our transmitted data must not exceed our Realtek RTL8188CUS-GR wifi module's limit of 300 Megabits per second at either 2.4 or 5 GHz frequency [9].

Quality of Streamed Video This requirement is still undefined, but is limited by the bandwidth of our wifi module, which mentioned above is 300 Megabits per second. We will try to maximize this video stream to improve the accuracy of our target tracking, which depends on high enough image quality. This means we need to optimize data efficiency of other communication and drone controls.

Power Consumption and Flight Time Our requirement for flight time is 10 minutes as a lower bound, and we picked a 5100mAh battery to achieve this. We calculate projected power consumption and overall flight time based on the following information:

- **Motors** 4 motors consuming 170W to lift 1kg (Model: AC 2830, 850 kV) [7]
- **Mass** 1282g for bare drone and battery and 300g for sensors and compute [7]
- **Battery** 5100mAH battery at 11.1V

Assuming we can only use 80% of the battery for safety reasons, we calculate the following [5]:

$$\begin{aligned}\text{Amp draw} &= \frac{170W}{1kg} * \frac{1282 + 300g}{1000g/kg} * \frac{1}{11.1V} \\ &= 24.23A \\ \text{Flight time} &= \frac{\text{capacity}}{\text{Amp draw}} \\ &= \frac{.8 * 5100mAH}{24.23 * 1000mAH} * \frac{60 \text{ min}}{H} \\ &= 10.10 \text{ min}\end{aligned}$$

Our flight time estimate is a lower bound since the compute and sensors will most likely not weigh 300g, and the 170Wkg assumes constant ascent (not hovering).

Data Storage Capacity High quality video will be stored on an external flash drive during flight. Given our captured video requirement of 720p, or 1280 x 720 sized RGB images and a frame rate of 30fps, a typical flight of 10 minutes would require the following data:

$$\begin{aligned}\frac{GB}{flight} &= \frac{\text{sec}}{\text{flight}} * \frac{\text{frames}}{\text{sec}} * \frac{\text{pixels}}{\text{frame}} * \frac{\text{Bytes}}{\text{pixel}} * \frac{\text{GB}}{\text{Bytes}} \\ &= \frac{15 * 60}{1} * \frac{30}{1} * \frac{1280 * 720}{1} * \frac{3}{1} * \frac{1}{10^9} \\ &= 50\end{aligned}$$

Where each pixel has 3 color channels for RGB, each ranging from 0 to 255 in value (1 Byte). In this case, we will need at least a 64GB card for video storage.

2.1 Scope

In order to meet these requirements within the time period, these are the simplifying assumptions we made:

- The target will be wearing a red CMU sweatshirt
- The drone will operate in open field without obstacles

- There will be WiFi access in test environment
- Daytime conditions with minimal wind
- The drone will be limited to tracking one person
- The target being tracked may be amongst 3 other people (not surrounded by a mob)
- Flight height no more than 20 feet

3 ARCHITECTURE OVERVIEW

Our system, which is shown in Fig. 3, is broken into three groups:

1. Drone compute
2. User compute
3. Wearable device

The drone compute contains all the hardware that will be on the drone to help navigate it. The camera on the drone will collect a live video feed and send it to an RPi through the MIPI Camera Serial Interface 2. The RPi will send the video it receives to the Jetson TX1 (referred as "TX1" from hereon), over WiFi using ROS, for processing.

The TX1 will use the video as input for the computer vision algorithm it runs to keep track of the position of the target and a state estimation algorithm which uses the target's position to predict the future path. Based on this, the TX1 will return the flight position commands back to the RPi over WiFi using ROS. The RPi will receive these position commands and forward it to the flight controller of the drone via USB. The flight controller will then be responsible to fly the drone to the correct position.

The video that the TX1 receives is sent to a 5" display via HDMI for the user to view. There are also buttons placed near the display for the user to interact with, which are processed by the TX1 through GPIO interface. The display and the buttons are powered by the TX1. The TX1 is powered by a rechargeable LiPo battery while the drone is powered by a 5100 mAh rechargeable battery. A high-level description of the division of computation between the important subsystems is as follows:

3.1 Jetson TX1

Used for video processing and processing user inputs via the buttons. The TX1 will use OpenCV library to process the captured video from the camera for the computer vision algorithm, and then feed the output of this into the state estimation algorithm. The implementation of the computer vision and state estimation algorithm will be discussed in detail in the System Description section.

3.2 RPi

Used as a middleman between the drone flight controller, camera and the TX1. Forwards updates in position from the flight controller to the TX1, new image frames from the camera to the TX1 and updated motion plans from the TX1 back to the drone flight controller.

3.3 Iris Drone Flight Controller

Takes high-level position/velocity/orientation motion planning waypoints that have been forwarded to it via the RPi and determines the speed of the motors required to achieve the desired path.

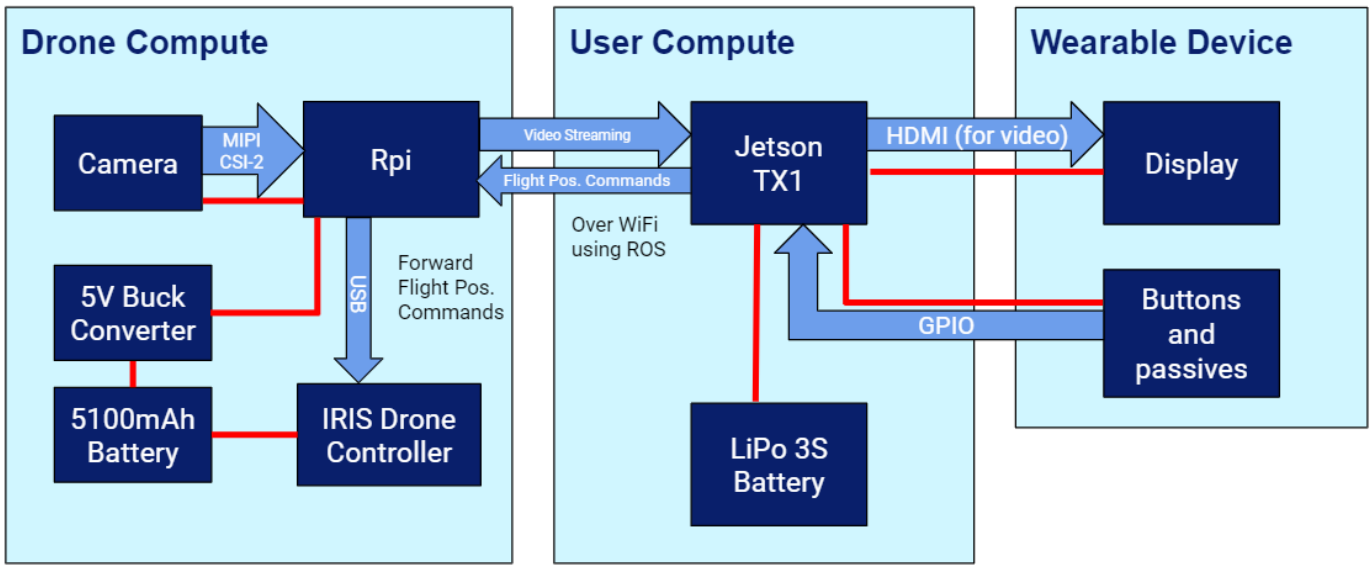

Figure 3: System block diagram

4 DESIGN TRADE STUDIES

4.1 On-Drone vs. On-User Compute

Originally, rather than have a Jetson TX1 located within the user's backpack and a RPi on the drone to interface with peripherals, we intended to just have a single Jetson Nano located on the drone that performed all the necessary software tasks. We figured that any extra compute on the user would be cumbersome and hoped to perform as much computation as possible on the drone itself. However, looking into the issue further, we realized that for our use-cases, most users would be carrying a backpack anyways and that performing computation on the drone severely limited us. The main points are:

1. **Power Consumption**

   Powering the Jetson Nano using the same battery as the drone would severely eat into the drones runtime given that the battery is only 5100 mAH. Considering that the drone by itself only runs for about 20-30 minutes on a single charge this would put us in danger of missing our flight time requirement. Not to mention that the additional weight of a Jetson Nano would force the drone to use up even more power to stay in flight. We would be forced to invest in a bigger battery, which once again would weigh the drone down even further.

2. **Peformance**

   Moving compute to the drone forces us to use a Jetson Nano rather than a TX1 (for size/weight reasons). However, a Nano contains only 128 GPU cores while the TX1 contains 256. For basic color filtering / blob detection this may be okay, but if we want to meet 15 fps requirement and have the flexibility of using more sophisticated approaches should we need to, the TX1 is a much safer approach. Here is a table [2] which benchmarks the Nano vs. TX1 for the same image processing task:

Algorithm and parameters / Jetson model	Jetson Nano	Jetson TX1
Host to Device	0.2	0.2
White Balance	0.6	0.32
HQLI Debayer	1.8	0.62
DFPD Debayer	4.7	2.4
MG Debayer	12.7	7.8
Color Correction with 3×4 matrix	1.7	1.05
Resize from 2K to 960×540	10.0	5.1
Resize from 2K to 1919×1079	19.8	9.0
Gamma (1920×1080)	1.4	0.96
JPEG Encoding (1920×1080, 90%, 4:2:0)	4.3	2.3
JPEG Encoding (1920×1080, 90%, 4:4:4)	6.8	3.1
JPEG2000 Encoding (lossy, 32×32, single mode)	81	70
JPEG2000 Encoding (lossless, 32×32, single mode)	190	180
Device to Host	0.1	0.1
Total for simple camera pipeline (ms)	**9.8**	**5.6**

Figure 4: Benchmarks for Nano vs. TX1

Of course, there also some benefits to handling compute on the drone rather than the user. Streaming the video down from the drone to the user over WiFi creates additional latency. Because of this, once the Jetson receives a frame, the user may have already moved slightly. However, given that we are sampling 15 times per second, we felt this would not be significant.

4.2 Wearable vs. Streaming to Computer

Originally, we intended to stream the live video feed to a computer. We thought this would be more convenient for us to implement since we would not need to buy anything extra or deal with hardware peripherals.

In the end, though, a small "wearable" display was decided to be used to show the streamed video from the drone rather than streaming the video to a computer. This is because in mission critical or vlogging applications, it is not user friendly to walk and hold a computer to be able to see the video from the drone. Further, for these applications having both hands available to perform other tasks than holding a computer is extremely important. Therefore, a 5" display will be strapped onto the user's arm, which will show the live video.

4.3 CV vs. Bluetooth Triangulation

As opposed to using computer vision to detect the target, we initially thought that we could use put a Bluetooth transmitter onto the target and position Bluetooth receivers around the environment to triangulate position. However, we decided against this for the following reasons:

1. **Target Detection**

   Bluetooth triangulation yields a very rough location estimate when the target is not limited to a 2-5 meter bubble. Current implementations on the market have 2 meter error even at distances as close as 6 meters [3]. The method is inherently noisy because any time the line of sight is blocked or the signal gets reflected off some surface, the estimate will be off. Since we do not want to restrict the user to a 2-5 meter space, we chose against using this method to track the user.

2. **Prior Knowledge**

   Between the three of us, we have little expertise in signals. In addition, our advisor is Professor Savvides, who's area of expertise is computer vision. Thus, we thought the computer vision approach would be more appropriate.

4.4 Buttons vs. Touch Screen

Buttons will be used to interact with the drone rather than a touchscreen display. This is because touchscreen displays are unreliable with the TX1, so buttons were decided on instead.

5 SYSTEM DESCRIPTION

Now that we have gone over the high-level design of our solution, here are the specifics on how each of the components will work and interact:

5.1 Onboard Compute

The drone itself has three broad categories of compute: flight controller, sensors, and RPi. The flight controller is a Pixhawk that uses the PX4 API with ROS to broadcast state of the drone and listen to motion commands. The Pixhawk can take a variety of control commands, from high level position (x, y, z) in local or global space to low level desired orientation in roll, pitch, and yaw. The Pixhawk has built-in PD controllers that control speed of the motors, and we only need to provide high level motion commands.

The Pixhawk already is built with several sensors for drone odometry: gyroscope, accelerometer, magnetometer (compass), and barometer. It also contains a GPS that provides global localization within 1-2 meters. We also already own a downward-facing Lidar Lite V3 rangefinder that provides accurate height estimate as well as an optical flow PX4Flow camera that points downwards. All of these sensors are fused together within the Pixhawk's internal Extended Kalman Filter.

Connected to the Pixhawk via UART, the RPi listens to these drone state estimates and forwards it down to the ground TX1. The RPi will also record video from an Arducam IMX477 MINI, store the 720p version on a flash drive, and stream down a lower quality video to the TX1 via ROS wifi. The TX1 will simultaneously publish motion commands to the RPi, which will forward these to the Pixhawk flight controller.

5.2 On-User Compute

The on-user compute is performed entirely on the Jetson TX1, which interfaces with the wearable display using HDMI and the buttons via GPIO. The control flow for the TX1's main() function is shown in Fig. 5. When the TX1 receives a start signal over GPIO, it will initialize ROS and send a "start" command to the RPi via ROS signalling the drone to rise to cruising altitude. Once this is performed, the TX1 launches three threads:

1. **Target Detection**

   This thread listens for new frames being sent from the RPi. Once a new frame arrives, it first sends it through the CV component to detect the (x, y) of the target as pixel coordinates and then adds this observation to the state estimator.

2. **Drone Management**

   This thread listens for updated drone state being sent from the RPi. It adds this drone state to the state estimator, then requests an updated motion plan from the state estimator, which is sent back to the RPi.

3. **Stop Detection**

   This thread waits for either a stop signal over GPIO or a "stop" command sent down from the RPi (in case the drone has determined it has to land). In the case of the former, the TX1 sends a "stop" command to the RPi, kills all threads and exits. In the case of the latter, the TX1 displays an error message on the display and continues.

In the next subsections, we will elaborate further on the computer vision and state estimation.

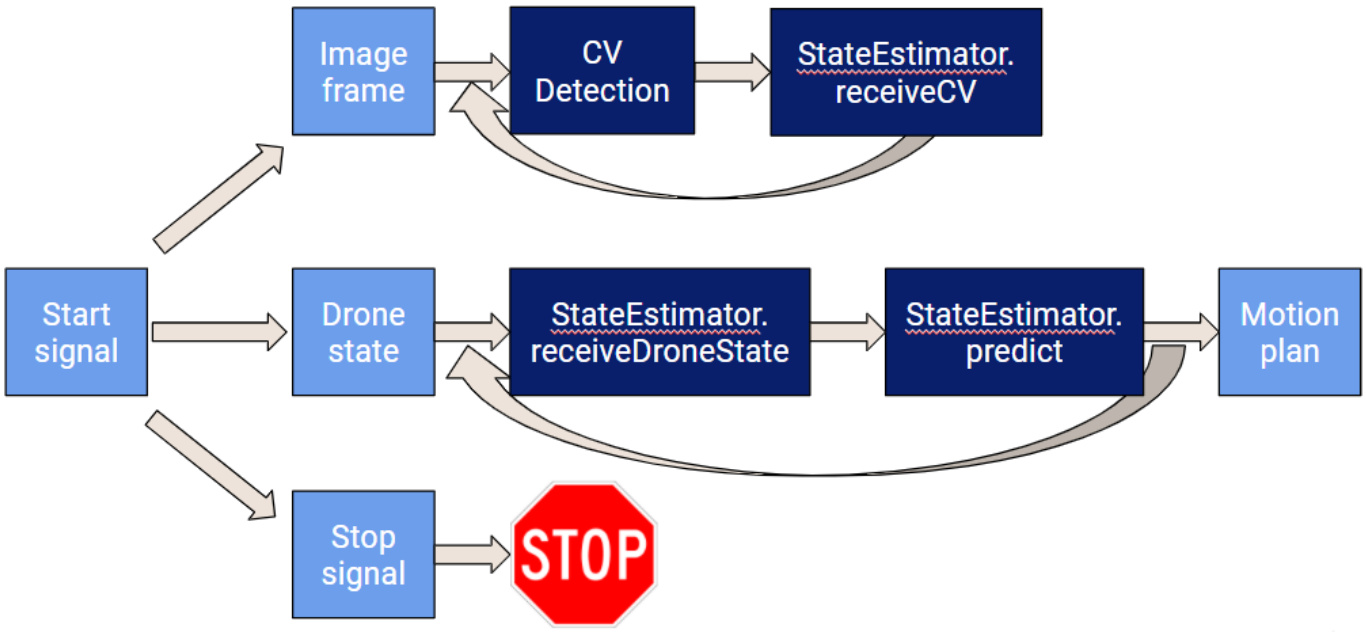

Figure 5: TX1 Control Flow

5.2.1 Computer Vision

The computer vision algorithm to detect the target reads frames captured by the camera and extracts the location of the target. To accomplish this, the captured frames go through the following stages:

1. **RGB Color Thresholding**

   The image read from the camera is an RGB image. In our scope, we mentioned we are tracking a target wearing a red colored shirt. Thus, the first step is to perform color thresholding on the RGB image to narrow down the red channel of the image. We predetermine the ranges for each of the 3 channels, blue(B), green(G), red(R) by manually adjusting them such that only the red pixels of the shirt would pass through. At first, we tried using HSV color space, but during testing, we realized it was too dependent on the lighting conditions. However, the RGB filter worked well in bright and dull lighting conditions.

2. **Noise Filter**

   The color thresholding by itself still allows some noisy pixels to pass through that are not the color of the shirt, which leads to the algorithm not detecting the target. To remove those, the frame is eroded using OpenCV which shrinks each group of pixels. This removes the noisy pixels that were getting past the color thresholding and affecting the detection of the target.

3. **Object Detection**

   Once the noise is removed, the OpenCV findContours function is used to detect the red shirt. Then, the moments function is used to find the center of the object detected. This becomes the (x,y) position of the target which will serve as an input to the state estimator.

5.2.2 State Estimation

For state estimation, we implemented our own Kalman Filter algorithm that models the user's state at all times in terms of:

- x/y position
- x/y velocity
- x/y acceleration
- estimated covariance between these parameters

The filter takes in two types of data to update this model:

- Pixel coordinates of the user's position each frame (sent from the CV)
- Updates in the drone's current state (sent from the RPi)

It is assumed that these data may be sent asynchronously, but will always be timestamped with an exact time that they were sent. The filter also maintains the following state transition matrices:

- **F**: Models how the user's state deterministically transitions from one period to the next. It assumes the user does not change acceleration and takes in Δt as a parameter.
- **Q**: Models random fluctuations in user state (due to user changing acceleration). Takes in assumed std. dev. in user accceleration as a parameter.
- **H**: Models how sensor data (the CV data and updates in drone state) relate to user's current state
- **R**: Models random fluctuations in sensor data (noisy observations). Takes in assumed std. dev. in observed user position as a parameter.

Using these matrices, the Kalman Filter probabilistically determines the most likely future state ($\hat{x}'_k$) and covariance in future state (P'_k) given the user's current state ($\hat{x}'_{k-1}$), current covariance in state (P'_{k-1}) and current sensor data ($\vec{z}_k$). Here are the equations for doing so [6]:

$$\hat{x}_k = \mathbf{F}\hat{x}'_{k-1}$$

$$P_k = \mathbf{F}P'_{k-1}\mathbf{F}^T + \mathbf{Q}$$

$$\mathbf{K'} = \mathbf{H}P_k\mathbf{H}^T(\mathbf{H}P_k\mathbf{H}^T + \mathbf{R})^{-1}$$

$$\hat{x}'_k = \hat{x}_k + \mathbf{K'}(\vec{z}_k - \mathbf{H}\hat{x}_k)$$

$$P'_k = P_k - \mathbf{K'}\mathbf{H}P_k$$

5.3 Wearable Device

A 5" display will be used to show the streamed video from the drone. The display has a 500mA current draw and can be powered by micro USB. The TX1 has an HDMI port and a micro USB that can supply the 500 mA current required by the display. Further, since this display has an HDMI port, the video will be sent through HDMI for display. This display is ideal as it is small, so it can be something we can wear on our arm, and requires no custom power management circuit as it can be powered through USB. Since right now we are planning to place the TX1 in a backpack and display on the arm, it is important to minimize the wiring between the two components.

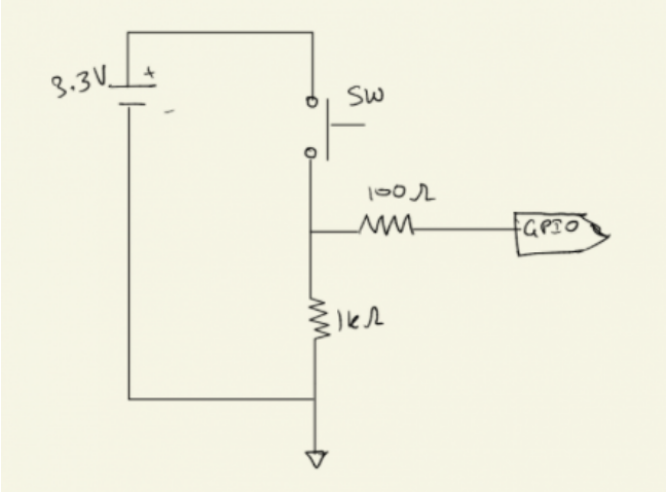

Figure 6: Button circuitry

Buttons will be used to interact with the drone. They will be assembled in a through hole package and will be placed near the display. We plan to have 4 buttons, one to start the drone flight, one to stop the drone flight, and two miscellaneous in case added functions are needed. The buttons will be powered by 3.3V GPIO (pin 1 on the J21 header on the TX1). Fig. 6 shows a simple circuitry to connect the stop button to the TX1. The same will be followed for the other three buttons.The buttons will be connected to J21 header pins 13, 16, 32, 33 on TX1 which all have a pull down network that will complete the circuit when the buttons are pressed.

6 PROJECT MANAGEMENT

6.1 Schedule

Our schedule (Fig. 7 in the Appendix) is composed of four phases: **design**, **pre-integration**, **integration**, and **performance testing**. Upon writing this report, we are nearing the end of our design phase. In design, we proposed various methods for building the system with various contingency plans in case certain ideas fail. We researched and purchased materials that would suite our requirements, and verified that already-owned materials would meet our requirements.

Our current step is pre-integration, which involves implementing and debugging various subsystems independently of one another. For instance, target detection can be implemented and tested using a fixed camera without involving the drone. Other tasks include target state estimation and prediction, drone motion planning in simulation, and the hardware communication protocol for the wearable display. After this three-week phase, integration involves combining all these independent subsystems together to produce a functional product. After this, performance testing will involve testing the system as a whole and recording performance through our various test metrics.

6.2 Team Member Responsibilities

Vedant Parekh will work on target detection in an image as well as the hardware communication protocols and circuitry for the wearable device.

Siddesh Nageswaran will work on target state estimation and prediction as well as design the housing for the TX1, display and drone's external hardware.

Alvin Shek will work on inferring 3D motion of the target from its 2D motion and the 2D image as well as generating drone trajectories from this information.

6.3 Risk Management

We have contingency plans in case our initial design doesn't meet our requirements.

1. **Noisy Target detection** During actual flight, target detection may be too noisy and unreliable if the image quality is too poor or the drone is too unstable. Drone acceleration and jerk will be minimized in trajectory optimization. If detection still isn't effective, we may buy a shirt with higher contrast, or even switch to other detection methods: April Tags [8] or CNN-based models that can be trained with domain randomization to be robust. We can also reduce the height of the drone's flight so the target is larger in the image.

2. **Noisy Target state prediction** Target state prediction may be too noisy if the target detection is noisy. We can bias more towards the historical model of target motion rather than frame by frame data if the incoming observations are too noisy. We can even set a threshold to throw away observations if they seem unrealistic. If the target suddenly "jumps" a large distance in pixel space that would translate to unrealistic motion in the real world, we can ignore this observation.

3. **Lost Sight of Target** The drone will continue to use the predicted trajectory of the target for a short amount of time, even if the target becomes occluded by some trees or obstacles. If the target is unseen for a prolonged duration, the drone will immediately fly back to home or land, and the user will receive a warning message on the display.

4. **Wifi Issues** As mentioned in the scope, we assume a strong wifi signal in the environment (on-campus). If the wifi connection ends up unstable during flight and motion commands do not stream at the minimum 2Hz, the drone will automatically land. If wifi issues persist, we can switch to bluetooth or radar as alternatives.

6.4 Budget

The bill of materials is shown in Fig. 8 of the Appendix. Any components that we plan to order through our $600 available budget are highlighted in yellow with their prices stated. For any components that are reused, we still stated their market price (although this does not count against our budget).

6.5 AWS Credit Usage

We would need use the AWS credits for experimenting with various deep learning approaches in the following tasks, the latter two which are post-MVP:

- **2D to 3D target motion** Possibly predicting 3D plane equations from 2D image and projecting target's 2D position onto the 3D ground plane. [10]
- **Advanced Target Detection** Possibly extend system to detect different targets with the tap of a button without needing them to wear a brightly colored shirt. [4]
- **Hand Gesture Controls** Human target uses arm gestures/motions to command drone to perform various sweeping camera motions.

References

[1] *12 Best Follow Me Drones And Follow You Technology Reviewed*. 2020. URL: https://www.dronezon.com/drone-reviews/best-follow-me-gps-mode-drone-technology-reviewed/.

[2] *Benchmarks for Jetson Nano, TX1, TX2 and AGX Xavier*. 2019. URL: https://fastcompression.medium.com/benchmarks-for-jetson-nano-tx1-tx2-and-agx-xavier-6c3b7105421d.

[3] *BLE Beacons for Indoor positioning – Beacon limitations*. URL: https://locatify.com/blog/ble-beacons-no-bull-beacon-review/.

[4] Michael Breitenstein et al. "Online Multi-Person Tracking-by-Detection from a Single, Uncalibrated Camera." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33 (Sept. 2011), pp. 1820–1833. DOI: 10.1109/TPAMI.2010.232.

[5] *Drone Flight Time Calculator*. 2018. URL: https://www.omnicalculator.com/other/drone-flight-time.

[6] *How a Kalman Filter Works, in pictures*. 2015. URL: https://www.bzarg.com/p/how-a-kalman-filter-works-in-pictures/.

[7] *Iris - The Ready to Fly UAV Quadcopter*. URL: http://www.arducopter.co.uk/iris-quadcopter-uav.html.

[8] E. Olson. "AprilTag: A robust and flexible visual fiducial system". In: *2011 IEEE International Conference on Robotics and Automation*. 2011, pp. 3400–3407. DOI: 10.1109/ICRA.2011.5979561.

[9] *USB Stick - WiFi Realtek 8188 (rtl8188cus)*. 2015. URL: http://domoticx.com/usb-stick-wifi-realtek-8188-rtl8188cus/.

[10] F. Yang and Zihan Zhou. "Recovering 3D Planes from a Single Image via Convolutional Neural Networks". In: *ECCV*. 2018.

Appendix

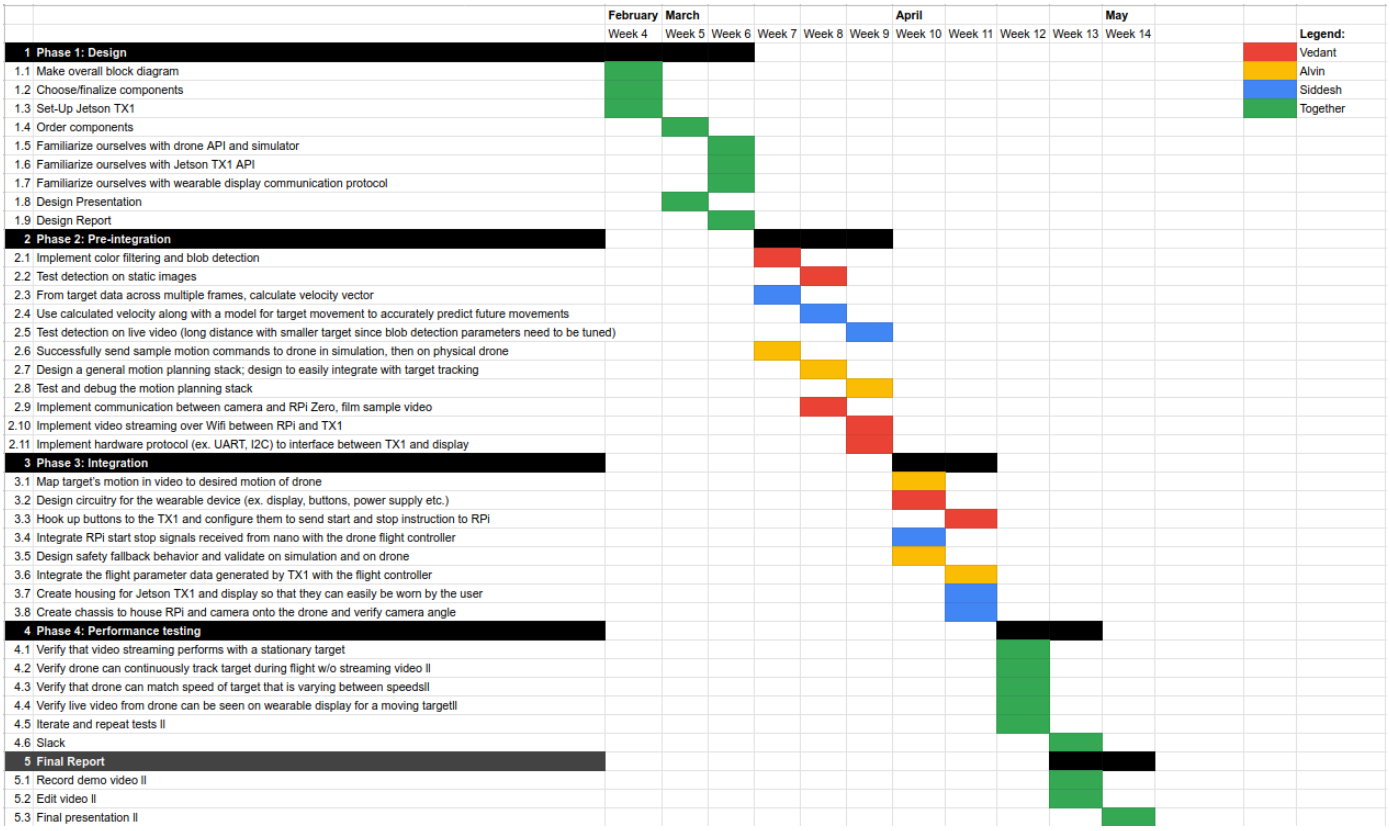

Figure 7: Gantt Chart Schedule

Material	Quantity	Provider	Total Price (if applicable)	Use
NVIDIA Jetson TX1	1	We Own	$300	Wearable Compute
JTAG Debugger	1	Order	$50	JTAG Debugger
3S 5100mAH 11.1v LiPo Battery	2	Order	$100	Battery for Wearable and Drone
SKYRC E3 Li-Po Battery Charger	1	We Own	$7	Battery for Wearable
LiPo Charging Bag	1	We Own	$9	Battery for Wearable
XT-60 male to 2.55mm barrel plug	1	Order	$7	Connect battery to TX1
Display	1	Order	$60	Display for Wearable
HDMI Cable	1	Order	$10	Connect Display to TX1
Micro-USB Cable	1	We Own		Power Display
Buttons	TBD	ECE Lab		Control Drone Start/Stop
Resistors	TBD	ECE Lab		Control Drone Start/Stop
Breadboard	1	ECE Lab		Mounting Buttons
Throughole PCB	1	ECE Lab		Mounting Buttons
Soldering Equipment	1	ECE Lab		Mounting Buttons
GPIO Jumper Wires	TBD	ECE Lab		Connect buttons to TX1
Copper Wires	TBD	ECE Lab		Connect buttons to TX1
Iris Drone	1	We Own	$270	The Drone Itself
Pixhawk Flight Controller	1	We Own	(included in Iris Drone)	Drone Control
Lidar Lite V3	1	We Own	$130	Drone Localization
PX4Flow Optical Flow Camera	1	We Own	$55	Drone Localization
Raspberry Pi 3 Model B+	1	We Own	$35	Microcontroller for Drone / Camera
Raspberry Pi Wifi Module	2	We Own	$14	Wifi for RPi
Buck Converter	1	We Own	$2	Connect RPi to Drone Battery
Camera	1	Order	$60	Drone Camera
Backpack	1	We Own		House the Wearable Compute
USB Keyboard	1	We Own		Programming TX1 / RPi
USB Mouse	1	We Own		Programming TX1 / RPi
Monitor	1	ECE Lab		Programming TX1 / RPi
3D Printer		ECE Lab		CAD Design
3D Printing and Laser Cut Material	1	Order	$100	3D Printing Material
Shipping Slack			$50	Slack for Shipping
Solidworks		Software		CAD Design
Robot Operating Systems		Software		Communications
PX4		Software		Drone Control
Total Cost			$1,259	
Total Spent			$437	

Figure 8: Bill of Materials