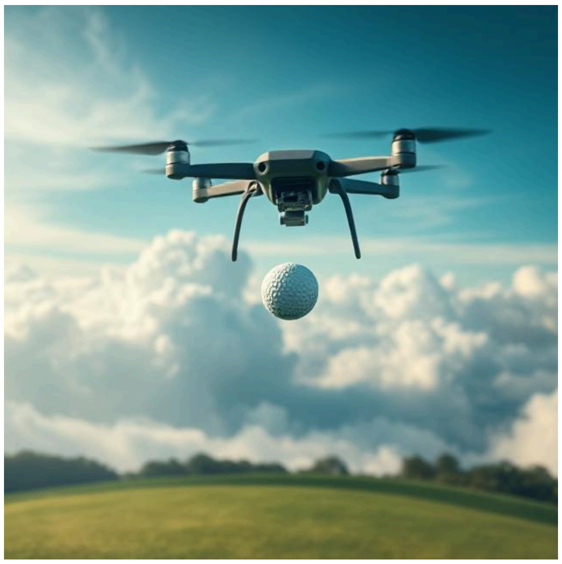
ROB498: Robotics Capstone Design
Team Final Report

PARSIGHT



Team #1: FLyRS

Felicia Liu (1006950042)
Luke Yang (1006959258)
Rohan Batchu (1006997181)
Siddharth Khanna (1006773341)

Abstract

Golf is an increasingly popular pastime among seniors, offering physical and mental benefits. However, many older golfers report difficulty tracking and locating the ball visually after hitting a shot due to age-related vision decline. Existing ball tracking tools focus on broadcasting graphics and post-game analysis, with no current solutions providing real-time visibility during game play.

ParSight is an autonomous drone-based, real-time golf ball tracking system designed to enhance visibility for senior golfers. The system maintains visual contact with the ball throughout its flight, flies stably, and makes fast, accurate, and robust tracking decisions in real time. To enable fast visual detection using its onboard RGB camera, we use a red golf ball and an HSV-based red filter to enhance red tones while suppressing blue and green channels, followed by added Gaussian blur to reduce image noise. The system uses OpenCV contour detection to identify red regions, scoring them based on size and roundness to select the most likely golf ball candidate. From the selected contour, the center point is extracted and used for tracking, which is performed through Image Based Visual Servoing, where pixel error, the offset between the ball's position on the screen and the image center, is transformed into a real-world pose correction. ParSight uses a PD controller to determine the drone's movement, enabling smooth, responsive flight without overshooting. Once the system detects the ball, ParSight autonomously takes off, ascends to tracking height, and locks onto its target ball. When the ball is launched (via a catapult proxy), the drone tracks its flight through rapid acceleration, airtime, bounces, and eventual rolling to a stop. It continues to hover over the ball's final position, providing constant visual feedback to aid users in locating it.

ParSight achieved a detection accuracy of 95% with only a 3% false positive rate. Although processing latency was just 30 ms, the total system reaction time was approximately 350 ms due to hardware constraints such as camera frame rate and resolution. As a result, the maximum positional error reached 75.51 cm, though the system could still reliably recover and reacquire the ball. These results demonstrated that ParSight met the objectives of the MVP: a working prototype that detects and tracks a golf ball quickly and fairly robustly in real-testing conditions.

The design and implementation of ParSight allowed the team to apply concepts from across their undergraduate journey while navigating the real-world challenges of robotics engineering, such as debugging hardware, tuning control systems, integrating computer vision pipelines, recovering from drone crashes, and collaborating effectively as a team throughout the process.

All source code, design files, and documentation related to this project can be found in our GitHub repository: https://github.com/liufeli2/FLyRS-ParSight-Capstone.

Table of Contents

Design Overview

To help senior golfers with declining vision, we present an autonomous drone-based real-time golf-ball tracking system. Using computer vision and control algorithms, ParSight can track and fly with a golf ball from the tee to its landing location, serving as a larger visual marker. Testing results demonstrate that our prototype reliably tracks a golf ball launched from a mini-catapult, meeting key performance metrics we have defined for a minimum viable product (MVP). With improved hardware, we are confident in our ability to extend this system to track full-speed outdoor shots.

With this solution in mind, we will now explain the significance of the problem it addresses. Golf has become an increasingly popular pastime among seniors, with participation among those aged 65 and older rising by 64% in the past decade and those 50 and above making up 43% of U.S. golfers in 2024 [1]. With the correlation between age and vision decline [2], senior golfers have expressed increasing challenges in tracking and locating the ball after hitting a shot [3][4]. Existing solutions primarily are for performance tracking, video enhancement, or post-game analysis:

- **Augmented Reality (AR) Golf Shot Tracers:** Enhances video recordings or live broadcasts by overlaying ball trajectories using AR, helping viewers follow the shot more easily. However, these systems are designed for post-processing or broadcast purposes and do not provide real-time tracking for the golfer on the course [5][6].
- **Manually Piloted Drones:** A dedicated pilot manually flies a drone to track shots, providing viewers with dynamic camera angles [7][8]. However, this setup is unrealistic for everyday golfing, as it requires constant human involvement and disrupts the natural flow of play.
- **Follow-Me Drones:** Drones equipped with the ability to autonomously track a subject for various sports applications such as skateboarding or biking [9]. These general solutions do not consider the following small objects travelling in the air at high speeds, such as a golf ball.

By considering the limitations of existing solutions, it is evident there is a gap in the market for a drone that autonomously tracks a golf ball in real-time and helps senior golfers locate their shots.

In developing our solution, the design process was guided by three core values: autonomy, safety, and creativity. First, restoring autonomy to senior golfers is a key element of our value proposition. By offering a larger visual marker to track the ball, our system helps restore the golfing experience seniors used to have without fundamentally changing how the game is played. Second, safety has been a central consideration in our engineering decisions, with an incremental approach that starts with basic implementations and tests and gradually progresses to more integrated evaluations. Finally, creativity shaped our ability to apply both past course knowledge and newly acquired skills in novel ways that have not been explored in this context before. Concerning our design process, the responsibilities and contributions of our team were also divided as follows:

- **Felicia:** Coded the red detection filter and entire tracking pipeline. Designed and initiated the incremental testing plan, beginning with testing and tuning the detection and tracking systems on a computer before porting them to the drone. Designed the landing gear in CAD.
- **Luke:** Set up development environment, deployment pipeline, and package management. Created development tools such as graphic user interfaces for easy debugging and automated deployment scripts. Performed data collection and analysis concerning evaluation metrics.
- **Rohan:** Established the RGB camera vision system. Developed the entire requirements model by researching testing metrics and ideal numbers for the final and MVP design.
- **Sid:** Developed the YOLO segmentation component before pivoting to the red detection filter, cleaning online datasets, collecting training data, and fine-tuning hyperparameters.

Background and Related Work

The following section covers all the relevant software frameworks we have used in our final design, explaining how they work, their general use case, and why we have chosen them for our final design solution. The three main components are: HSV colour detection, Image-Based Visual Servoing (IBVS) and Proportional Derivative (PD) Control.

HSV Colour Detection

Colour detection algorithms identify pixels in an image that match a specific colour or colour range [10]. The HSV colour space is often preferred over RGB because it separates hue, saturation, and value, making colour detection more robust and reliable [11]. The algorithm is particularly well suited in industries like manufacturing, automotive, and food processing where distinguishing between objects based on colour is essential. For example, identifying similar-looking products, inspecting colored adhesives or baked goods, and detecting naturally colourful materials [12]. We selected HSV colour detection for our design because golf balls are typically monochromatic, aligning well with the algorithm's strengths. Additionally, HSV-based detection is computationally efficient, which is critical for tracking fast-moving objects like a golf ball.

Image-Based Visual Servoing

Image-Based Visual Servoing (IBVS) is a common practice in robotics that utilizes computer vision to provide a closed-loop position control for a robot end-effector. Specifically for mobile robotics, the goal is to control the vehicle's pose with respect to some landmark [13]. Within industry, visual servoing has been used for bin picking, packaging, and sorting tasks in a controlled environment [14]. We chose IBVS for our design because it aligns well with our design problem. Specifically, we control the drone's pose relative to the golf ball, which acts as a visual landmark in the camera frame. Compared to dynamics-based control systems, IBVS also offers a simpler implementation that can take advantage of the drone's existing integrated setpoint control system.

Proportional Derivative Control

Proportional Derivative (PD) control is a simplified form of the widely used Proportional Integral Derivative (PID) control algorithm. By measuring input data from sensors, the difference between the actual value and the desired setpoint of a system can be calculated and used to adjust the outputs to correct the error [15]. Industry use cases include temperature, power electronics, automatic machines, and automobile control [16]. We chose to use PD control when determining the desired setpoint from the IBVS control output because of its reputable industry performance. We excluded the integral component to prioritize rapid response and effective damping of oscillations, as steady-state accuracy was not a primary concern for our application.

Design Objectives and Solution Details

In the submitted proposal and during the early stages of the project, the requirements were not as well defined; we have since fleshed out the requirements model through further development. The new requirements are better defined to enable benchmarking and testing of our prototype. In addition to this, feedback from course instructors informed our new scope, which constrained our primary functions to tracking the golf ball, hovering over the golf ball, and being easy to set up and use. One of the goals we removed was the obstacle avoidance due to the limited time frame for prototyping, as well as the limited hardware capabilities of the drone. Some functions were also altered and added to better describe the system's interfaces and necessary capabilities that were realized during development. Additionally, performance requirements were developed and altered as well as including more finalized specifications.

Our project scope now has 3 main objectives with the following requirement codes:

- MLR-001: Tracking the golf ball from tee-off to rest
- MLR-002: Become a visual marker for the ball at rest
- MLR-003: Ensuring easy setup and use of the drone

These objectives are further broken down into system-level requirements (SLRs) and subsystem requirements, with the following system and subsystems and their codes:

- **SLR** - System Level Requirements
- **IMG** - Sensing and Perception
- **OBD** - Image Processing and Ball Detection
- **FLC** - Tracking and Path Planning

ParSight employs a requirement classification of three types to use as a framework for aligning design decisions with system performance and operational constraints. The category types are Functional (F), Performance (P), and Constraint (C). The requirement codes follow the syntax: *<System>-<Category Type>-<Number>*.

As development continued, we found that we were working in three primary areas of imaging, object detection, and actuation. As such, these 3 areas were made into subsystems (IMG, OBD, FLC) that interact with each other and other hardware onboard to create an effective solution as seen in *Figure 1*. The Localization system is the RealSense/Vicon system that provides real-world position data to send setpoints to the flight controller.

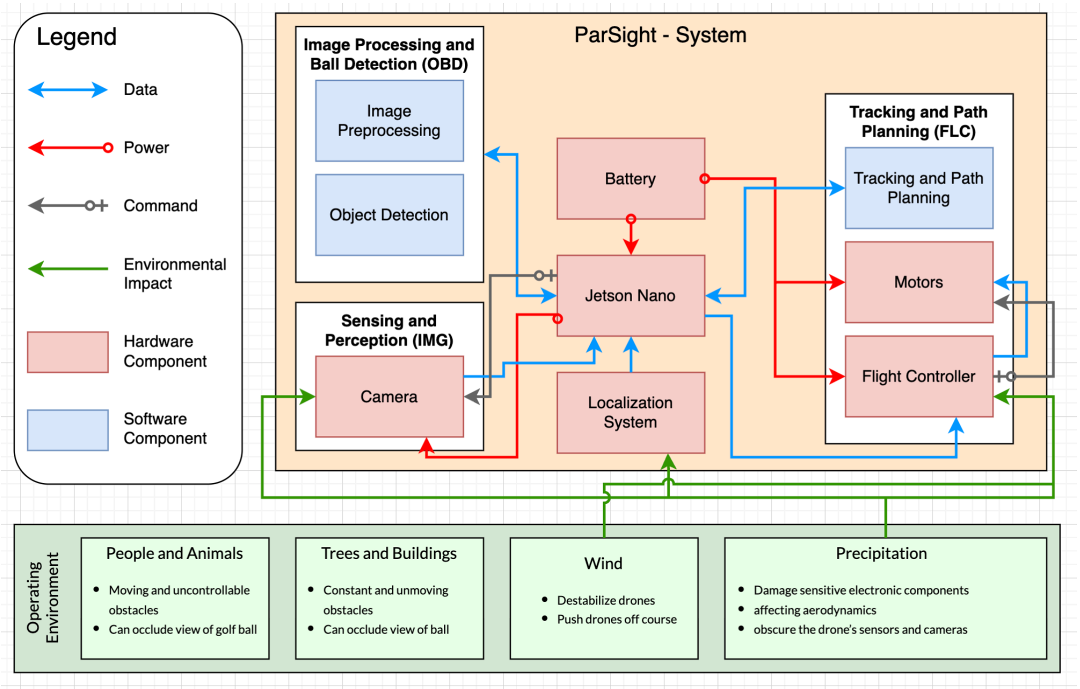


Figure 1. *System Block Diagram showing the connections and interfaces between subsystems, the system, system hardware, and the environment.*

Table 1. *System Level Requirements (SLRs).*

Code	Requirement	Explanation	Parent Req
SLR-P-001	The processing latency from image to motor commands shall be less than 100 ms	The software pipeline from perception to planning must occur in less than 100ms to track a golf ball with the current hardware specifications [1]	MLR-001
SLR-P-002	The drone shall be ready to use in under 10 minutes	To reduce technical issues and account for the varying technical skills of users, ease of use is necessary	MLR-003
SLR-F-001	The drone shall maintain a view of and track the ball throughout flight	To track the golf ball effectively, the camera must have the golf ball in frame at all times	MLR-001
SLR-F-002	The drone shall have a safe and stable flight when tracking the ball	Drone stability provides smoother transitions for planning and will require less image processing	MLR-001
SLR-F-003	The drone shall have all the computation onboard	Due to the high possible speed of the golf ball, it is not viable to have a delay in receiving and sending data to a remote computer	MLR-001
SLR-F-004	The drone shall detect the ball's position from overhead	Since the ball starts on the ground, tracking from overhead will provide the best possible view of the ball throughout flight	MLR-001
SLR-F-005	All drone power shall be sourced from the onboard battery	The far distance travelled by the golf ball makes a wired drone unviable	MLR-001
SLR-F-006	The drone shall maintain a distance such that it will have minimal impact on the golf ball's trajectory	A goal of the project is to minimize the effect of the prototype on the sport of golfing, therefore, the drone must be far enough away such that the lift generated by the drone has minimal effect on the golf ball	MLR-001
SLR-F-007	The drone shall hover over the golf ball at rest	To provide a larger visual marker for the golf ball location after a swing	MLR-002

Table 2. *Subsystem Level Requirements.*

Code	Requirement	Explanation	Parent Req
SENSING AND PERCEPTION REQUIREMENTS			
IMG-C-001	The camera shall weigh less than 500 grams	For the drone to be able to fly and react to control inputs, the drone cannot have too much added weight	SLR-F-002
IMG-P-001	The camera shall provide images at least 20 Hz to the Jetson Nano	For effective tracking, based on the calculations, we require 20 Hz to capture ball motion off of tee-off	SLR-P-001
IMG-P-002	The camera shall send images with a resolution of at least 480p	Image quality determines the ability of the object detection algorithm to find the ball in the image. A higher quality is generally better, but to find the features in the testing environment, 480p should be enough, as the detection is at a lower distance and fewer overall features [17]	SLR-F-001
IMG-F-001	The camera shall be securely attached to the drone, facing the ground throughout flight	If the camera is not securely attached, the image and information gathered will not be effective for tracking the golf ball at all times	SLR-F-004
IMG-F-002	The camera shall operate on power provided by the Jetson Nano	The Jetson provides power for all external devices and sensors, and as such, must be able to run on the provided power	SLR-F-003
IMG-F-003	The camera shall provide imaging data directly to the Jetson Nano through onboard connections	The Jetson is the onboard computer, and to achieve the objectives, it must receive images from the data	SLR-F-003

[1] Based on a camera FOV of 60°, estimated ball speed of 10m/s and a flight height of 2.3m, the distance the ball could travel before being lost from the frame is 1.328m which the ball travels in about 0.1328s. This requires that the pipeline speed be capable of detecting and reacting to this at a rate of at least 7.5 Hz for at least one detection.

IMG-F-004	The Jetson Nano shall provide imaging data to any required programs	The drone and all controls are being run through ROS2, and as such, if the pipeline requires multiple scripts, the images may need to be available in all files when being run all at once	SLR-F-003
IMAGE PROCESSING AND OBJECT DETECTION REQUIREMENTS			
OBD-P-001	The object detection algorithm shall operate at a frequency of at least 20 Hz	For effective tracking and based on the estimated ball speed of the MVP, the minimum required frequency was decided	SLR-P-001
OBD-P-002	The object detection algorithm shall have a detection true positive rate of at least 90%	Effective tracking requires high recall, so when there are golf balls in the frame, the algorithm can pick out the golf ball and can identify when there is no ball in frame [18]	SLR-F-001, SLR-F-002
OBD-P-003	The object detection algorithm shall have a False Positive Rate of less than 5%	When detecting the golf ball, detecting other objects in the image background would make tracking unviable [19]	SLR-F-001, SLR-F-002
OBD-P-004	The object detection algorithm shall have a true positive rate of at least 85% in bright, normal, and low-light conditions	Golf can be played in all sorts of lighting conditions, so ensuring that the performance in varying environments is essential	SLR-F-001, SLR-F-002
OBD-F-001	The object detection algorithm shall receive imaging data from the imaging system	The object detection algorithm requires images to identify and locate the golf ball	SLR-F-003
OBD-F-002	The object detection algorithm shall segment the golf ball out of the received image	To achieve project objectives, the object detection algorithm is required to identify and locate golf balls in the image	SLR-F-001
TRACKING AND PATH PLANNING REQUIREMENTS			
FLC-P-001	Using the object detection output, the flight control shall determine drone motion within 20ms	Tracking the estimated ball position in the image and determining the next drone position needs to be done within latency to ensure the ball stays in frame	SLR-P-001
FLC-P-002	The flight control shall maintain a distance of at most 20 cm from the ball's resting position	The goal of the project is to provide a better location of the ball's final position for the elderly and provide a larger target	SLR-F-007
FLC-P-003	The flight control shall move with the ball within 150 ms of motion	To reduce latency and ensure that the golf ball is within the camera frame	SLR-F-001
FLC-P-004	The flight control shall move such that the gold ball is within 10 pixels of the center of the image	To ensure tracking precision throughout flight, a maximum image frame pixel distance must be maintained	SLR-F-004
FLC-P-005	The drone shall maintain a height of 2.3m above the ball to limit alteration of the golf ball trajectory	To minimize the effects of wind on the trajectory of the ball, a height of 2.3m was chosen to balance Vicon performance and lift force	SLR-F-006
FLC-F-001	The flight control shall stay within the boundaries of the golf course	To ensure safety for others and the drone, it should remain within a specified region (ie. golf course)	SLR-F-002
FLP-F-002	The flight control shall have a max speed equal to or greater than the golf ball speed	If the drone is slower than the ball, tracking will not be effective	SLR-F-001, SLR-F-002
FLC-F-003	The flight controller shall control the drone motors to achieve the motion specified by the FLC	Conversion between the image frame setpoint and the actual drone motion is essential for tracking	SLR-F-001, SLR-F-002
FLC-F-004	The flight controller shall land the drone if the battery is at 10% or less	This allows for drone safety to be maintained and allows for a controllable landing sequence that will not damage the environment	SLR-F-005

Design Prototype Overview

This section outlines the implementation of ParSight, including the design and final functionality of the prototype developed. We describe the system architecture and the plan-sense-act paradigm used, including both the hardware and software components that enable our real-time golf ball tracking. We also explain the prototype's core mechanisms for ball detection, tracking, and flight control to illustrate how the system operates end-to-end. Finally, we discuss the extent to which the current prototype meets the project's primary objectives, along with the key considerations and modifications needed to scale ParSight for real-world deployment.

<u>Sense-Plan-Act Robotics Paradigm</u>
ParSight follows the Sense-Plan-Act model, a foundational paradigm in robotics that structures a robot's behaviour around three key functions: sensing the environment, planning a response, and acting upon that plan. This model provides a clear and modular way to break down the operations of ParSight's autonomous golf ball tracking system, helping us understand how each hardware component contributes and integrates into the overall functionality of the system at a high level.

Sensors (Input)
The primary sensor used in ParSight is a downward-facing RGB camera that continuously captures a live video feed of the environment beneath the drone, where we expect to see the golf ball. We selected the NexiGo N660 USB webcam, a practical solution for our vision-based tracking needs [20]. This camera can capture video at 30 frames per second, which is suitable for our proof-of-concept tracking tasks. It features a wide-angle 60° field of view, allowing us to observe a broad portion of the scene and track the golf ball effectively from the air. The 1080p Full HD resolution (1920x1080) provides more detail than strictly necessary, but the extra resolution offers flexibility, allowing us to downsample frames to reduce processing load as needed. Its USB-A connectivity simplified hardware integration, as our Jetson experienced issues with the CSI connection port. We mounted the NexiGo camera in a downward-facing orientation, fixed on the drone to ensure that its view was aligned with the drone's local frame of reference. Once visual input is captured, image processing techniques are used to isolate the location of the red golf ball within the frame, informing the planning system on how the drone should respond.

Figure 2. *Sensing Hardware: NexiGo N660 webcam.*

Planners (Onboard Computation)
The onboard planning computation is handled by an NVIDIA Jetson Nano, a compact embedded computer with a quad-core ARM Cortex-A57 CPU and a 128-core Maxwell GPU capable of 0.5 TFLOPs (FP16) provided by the course instructors. This processing unit runs the Robot Operating System 2 (ROS2), which manages communication between sensing, planning, and actuation components in a modular and distributed manner. During planning, ParSight processes the image coordinate data to calculate the pixel offset between the detected golf ball and the center of the frame. This offset serves as the input to an Image-Based Visual Servoing (IBVS) algorithm, which

determines the directional correction needed to maintain visual alignment. We designed our own Proportional-Derivative (PD) controller external to the built-in Orange Cube + Flight Controller in order to dynamically impose position-based control, which computes the appropriate control signals based on this error, balancing responsiveness with stability. These control outputs are subsequently passed to the actuation system to update the drone's position in real time.

Figure 3. *Planning Hardware: Jetson Nano (IBVS and PD control).*

Actuators (Output)
Once a corrective plan is generated, the system translates it into updated flight setpoints that define the drone's desired position in real time. These setpoints are published through ROS2 on the Jetson Nano and transmitted to the Orange Cube+ flight controller, which interprets them and initiates the appropriate motor commands. The flight controller adjusts the drone's motion by regulating the Electronic Speed Controllers (ESCs), which in turn drive the brushless motors to produce the necessary thrust and orientation adjustments. ParSight is built on the Minion Mini H-Quad racing drone platform, equipped with 5" three-blade propellers, Spedix ES30 HV30A ESCs, and Rotorgeeks 7075 Series 2206 2350KV brushless motors. The Orange Cube+ flight controller, featuring a 32-bit ARM Cortex M7 processor and integrated IMUs, ensures precise and responsive navigation. This tightly integrated actuation system enables fast, stable flight, allowing ParSight to follow the golf ball's movement and maintain visual lock throughout its trajectory.

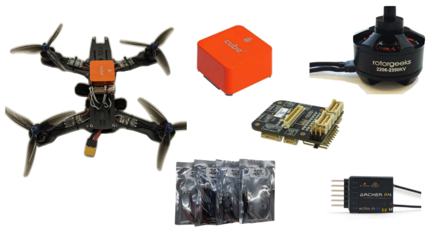


Figure 4. *Flight and Actuation Hardware: Minion Mini H-Quad drone with 5" three-blade propellers, Spedix ES30 HV30A ESCs, Rotorgeeks 2206 2350KV brushless motors, and Orange Cube+ flight controller receiving ROS2 setpoints.*

<u>End-to-End Prototype Pipeline</u>
Now we'll walk through, in more detail, each step of the ParSight prototype pipeline, highlighting how the system operates from input to output (end-to-end). This includes the flow of information from sensing and image capture, through onboard planning and decision-making, all the way to motor-level actuation. Supporting images illustrate each stage of the pipeline, giving a full picture of how the components work together to enable real-time autonomous golf ball tracking.

Sensing & Perception
The first step in ParSight's end-to-end pipeline is visual sensing. We developed a dedicated ROS 2 node that continuously captures, downsizes, and publishes webcam images, which form the core of ParSight's sensing pipeline. To optimize performance and reduce computational load, each captured frame is resized to 256×256 pixels, balancing image quality with processing efficiency.

The resized frames are then converted into ROS Image messages using CvBridge, ensuring smooth integration with the ROS environment. To maintain a consistent flow of image data, a 30 FPS timer triggers the *timer_callback()* function, ensuring frames are captured and published at a steady rate. This enables the */camera/image_raw* topic to receive real-time image data from the camera.

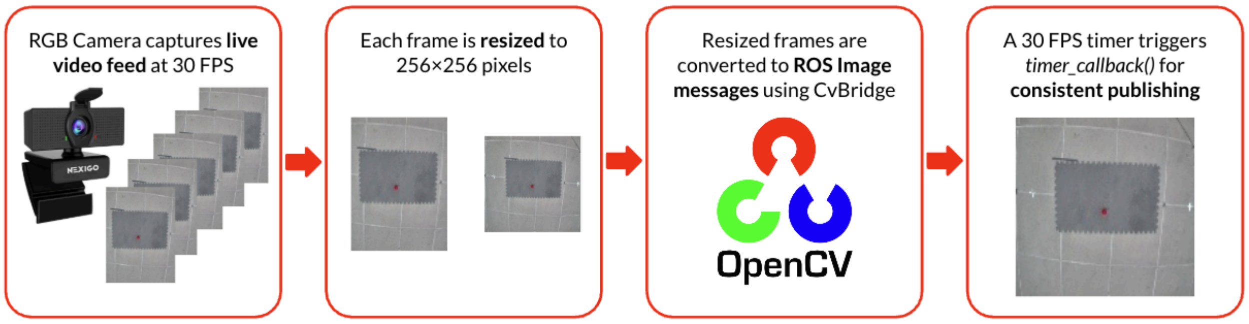

RGB Camera captures **live video feed** at 30 FPS
Each frame is **resized** to 256×256 pixels
Resized frames are converted to **ROS Image messages** using CvBridge
OpenCV
A 30 FPS timer triggers *timer_callback()* for **consistent publishing**

Figure 5. *Image Acquisition Pipeline: Webcam frames resized to 256×256 and published at 30 FPS via ROS2.*

Image Processing

The goal of our processing pipeline is to clean up the frames for effective ball detection. We chose a red golf ball for testing due to its strong contrast against the surrounding MY580 environment, making it easier to detect. As a result, the image processing is specifically designed to target the detection of this red ball. The raw camera frames are first received from the */camera/image_raw* topic as ROS Image messages and then converted into an OpenCV-compatible format. The frames are subsequently transformed from the standard RGB (Red, Green, Blue) colour space to HSV (Hue, Saturation, Value), as HSV provides better robustness to lighting changes and allows more precise colour segmentation. Next, a mask is created to assist in identifying the red golf ball by applying colour thresholds specifically tuned for the red hue. To minimize interference from other colours, separate masks for green and blue are generated, inverted, and combined with the red mask to focus solely on the ball. A Gaussian blur is then applied to smooth edges and reduce small noise artifacts from the floor texture, enhancing the detection's stability. This results in the final processed image, which is now clean and ready for ball detection. Although techniques like image undistortion, histogram equalization, and contrast adjustments were considered, they were not implemented in our final prototype. Because our camera performed well without noticeable distortion, even at the edges, and the ball's circular shape was always clearly detectable, adding these extra processing steps would have slowed down the system, which needed to maintain fast computation times for real-time performance. Hence, we chose a minimal image cleaning approach, prioritizing efficiency and simplicity.

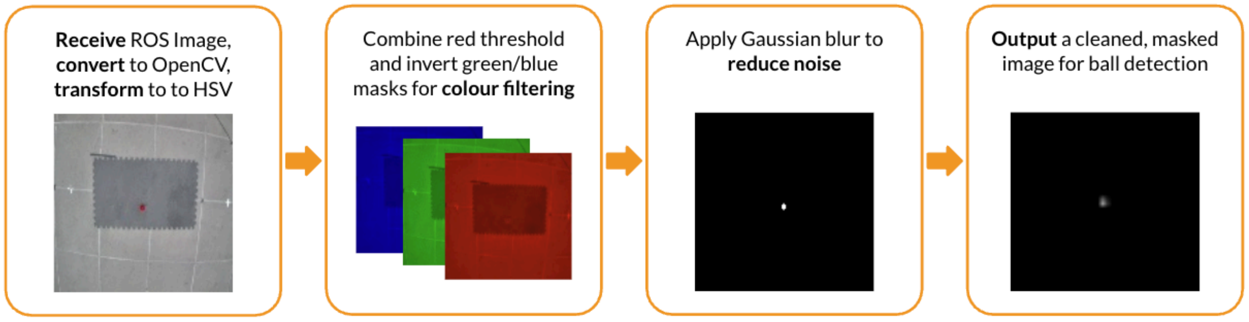

Receive ROS Image, **convert** to OpenCV, **transform** to to HSV
Combine red threshold and invert green/blue masks for **colour filtering**
Apply Gaussian blur to **reduce noise**
Output a cleaned, masked image for ball detection

Figure 6. *Image Processing Pipeline: Raw images are converted to HSV, masked for red, cleaned with Gaussian blur, and prepared for ball detection.*

Ball Detection

Once the image has been cleaned and processed, we apply contour detection to identify the red golf ball within the frame. Contours are detected in the blurred mask, which helps outline the edges of potential targets. To ensure robustness in selecting the correct contour, we implemented a scoring system that evaluates the size and roundness of each detected contour. Size is calculated as the area of the contour, while roundness is determined by comparing the radius derived from the perimeter to that calculated from the area. A perfect circle will have a roundness score close to 1, while non-circular objects will exhibit greater deviation from 1. By scaling the size by the roundness score, we can determine the most likely candidate for the ball, prioritizing the largest and most round contours. Once the best contour is selected, its center is calculated using image moments. If no contours are found, the system defaults to calculating the center of mass of the entire binary mask as a fallback. This center point, which represents the location of the ball in the frame, is then passed on to the tracking system. If no ball is detected in a given frame, no update is given, and the system assumes that the ball remains stationary. Note in this step that while our system is flexible, allowing us to switch the target ball colour by adjusting a single parameter, the detection method still relies on colour filtering. The ball must be a distinctly different colour from its surroundings and the largest and roundest object of that colour in the environment. While red is ideal for our test environment, white balls, which are more commonly used in real-world outdoor golf settings, would contrast well against green grass, allowing our method to still work.

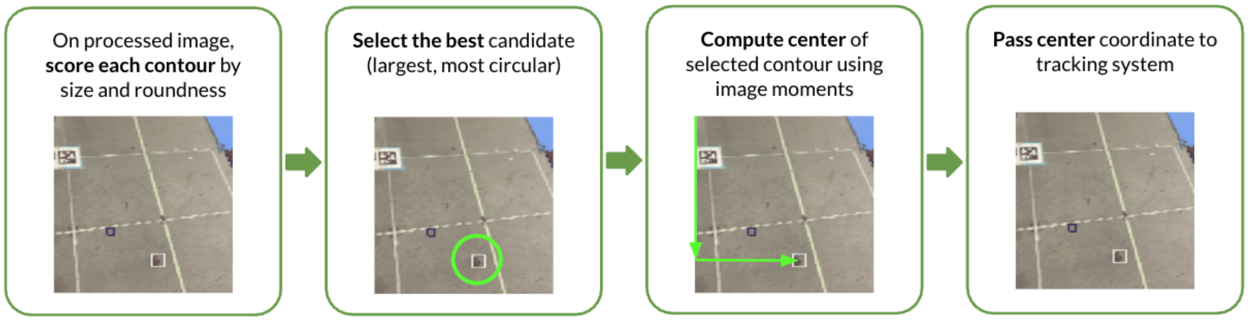

On processed image, **score each contour** by size and roundness
Select the best candidate (largest, most circular)
Compute center of selected contour using image moments
Pass center coordinate to tracking system

Figure 7. *Ball Detection Pipeline: Contours in the red mask are scored for size and roundness; the best match's center is used for tracking.*

Tracking Control

Once the image plane center point of the ball's position is obtained, the system transitions into the tracking phase. ParSight uses a localized approach, making incremental movements toward the ball at each time step, continuously monitoring the ball's position and adjusting the drone's position accordingly, enabling it to follow the target. This is accomplished using an IBVS method, which allows the drone to dynamically respond to the ball's changing location within the frame. We selected a PD controller for this purpose, as it provides a good balance between responsiveness and stability, allowing fast correction of positional errors while reducing oscillations and overshoot, thereby enabling smoother and more accurate tracking of the golf ball. The IBVS pipeline begins with calculating pixel position errors in the X and Y directions. This is done by determining the difference between the ball's position in the frame and the frame's center, giving a pixel error vector. If this vector's magnitude is within a small threshold (5 pixels), the drone maintains its position and hovers in place. If the error exceeds the threshold, the drone proceeds to compute an error and apply a correction. To translate the pixel offset into a real-world movement, we use a fixed calibration based on the fixed flying height, fixed frame resolution, and the known tile sizes in the world seen in the image frame. Next, PD control is applied. The system calculates the time difference between the current and previous frames to determine the rate of change in error (derivative component). Proportional error is multiplied by K_p gain, and the derivative of the error is multiplied by a K_d gain. The sum of these two components yields the required positional adjustment. These gain values are tuned based on real-world testing, accounting for factors like drone stability, damping, and overshoot. The resulting control values for movement in the X and Y directions are used to update the drone's setpoint position continuously.

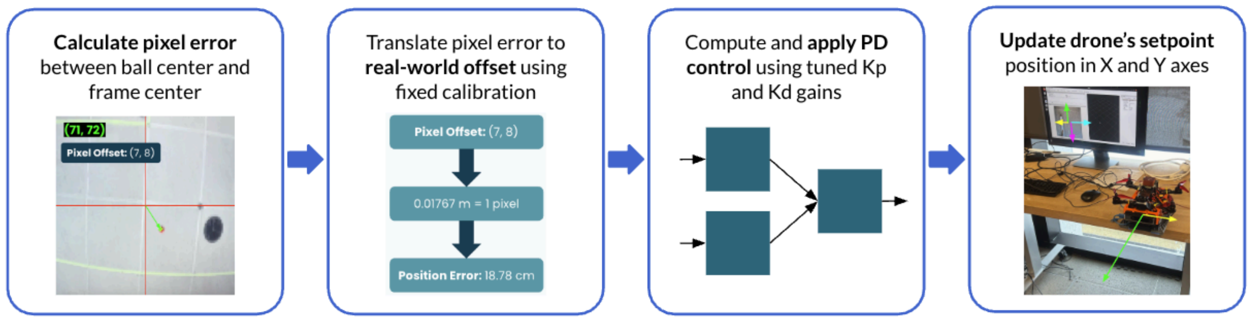

Calculate pixel error between ball center and frame center
(71, 72)
Pixel Offset: (7, 8)
Translate pixel error to real-world offset using fixed calibration
Pixel Offset: (7, 8)
0.01767 m = 1 pixel
Position Error: 18.78 cm
Compute and apply PD control using tuned Kp and Kd gains
Update drone's setpoint position in X and Y axes

Figure 8. *Tracking Control Pipeline: IBVS with PD control adjusts the drone's setpoint based on pixel error between ball and image center.*

Actuation
Actuation of the tracking commands is handled through the integration of ROS2 and the drone's onboard flight control system. The vision pose and computed setpoint positions generated during the tracking process are published via ROS2 on the NVIDIA Jetson Nano. These commands are transmitted to the Minion Mini H-Quad racing drone, where the Orange Cube+ flight controller interprets them and actuates the appropriate motor outputs. The flight controller manages the Electronic Speed Controllers (ESCs) and motors, translating the desired setpoint positions into precise physical movements, allowing the drone to follow the golf ball in real time.

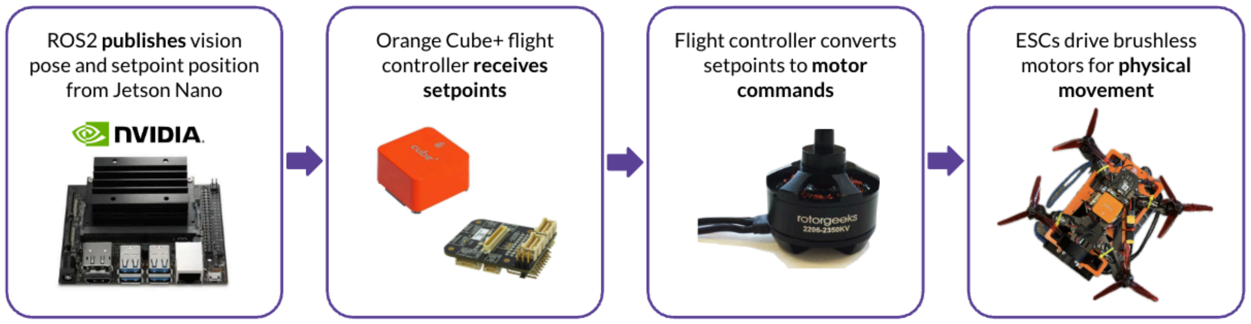

ROS2 publishes vision pose and setpoint position from Jetson Nano
NVIDIA
Orange Cube+ flight controller receives setpoints
Flight controller converts setpoints to motor commands
rotorgeeks 2206-2350KV
ESCs drive brushless motors for physical movement

Figure 9. *Actuation Pipeline: ROS2 setpoints guide motor outputs via the Orange Cube+ flight controller to follow the golf ball.*

<u>Evaluating Prototype Success Against Objectives</u>
While the later report sections will walk through testing results in detail, it's worth first discussing the higher-level design choices that guided our prototype and how these align with our objectives.

Successes
Our approach prioritized fast and responsive tracking, which is why we selected colour-based detection as it's lightweight and computationally efficient, which suited our real-time constraints. The amount of image processing we used was also deliberately minimal, enough to sufficiently isolate the red ball effectively while avoiding additional processing steps that would slow the pipeline down without adding much benefit. For tracking, we opted for a reactive PD controller

over more complex solutions like Model Predictive Control (MPC). This allowed us to maintain speed and responsiveness without needing to solve optimizations. In terms of detection accuracy, our chosen scoring system, based on both contour size and roundness, added robustness. For example, in initial tests, red tape on the MY580 floor sometimes created false positives because, from a downsampled view, the red tape and the ball appeared similar in size. We observed that their roundness scores were significantly different, allowing us to distinguish the ball reliably.

Challenges
One key limitation we encountered was in tracking high-speed ball motion, particularly during sudden launches. In real-world behaviour, a golf ball doesn't gradually build up speed, it accelerates almost instantaneously from rest to high velocity. This rapid motion poses a significant challenge for our system as the ball often moves so quickly that it exits the image frame before the drone has a chance to react. This delay is primarily due to two factors: the frame rate at which the camera can capture and send images, and the total time required for the rest of the image processing and tracking pipeline to complete its tasks. Another contributing factor is the resolution of the images we use. While higher-resolution frames provide more detail for detecting the ball accurately, they also slow down the processing and the rate at which we can publish raw image updates. This trade-off between precision and speed becomes critical in scenarios involving fast-moving objects. As a result, while our system performs well for moderate-speed tracking and maintains accuracy in controlled environments, further work is needed to improve its responsiveness to rapid motion.

<u>Scaling Toward Real-World Deployment</u>
To transition our ParSight prototype into a real-world system, several upgrades and optimizations would be helpful. A key improvement would be incorporating a higher-frame-rate camera capable of capturing fast-moving objects more reliably. This would reduce the likelihood of missing the golf ball during sudden launches, which currently outpace our system's ability to react in time. A more powerful onboard hardware, such as the Jetson Orin Nano, would also allow us to process higher resolution frames without compromising speed [21]. Our current pipeline becomes noticeably slower beyond a frame size of 256×256 pixels, limiting our detection precision. More capable computing resources would also allow us to experiment with more intensive techniques such as MPC, enabling the system to anticipate movement rather than solely reacting to it, or consider performing ball detection using YOLO and other deep learning models. Another important area for scaling is more robust false-positive rejection. While our roundness and size-based scoring system has been effective, integrating additional cues like temporal consistency or trajectory validation could reduce misclassification from similarly colored artifacts in complex environments. Together, these enhancements would help narrow the gap between our prototype and a deployable real-time solution suitable for dynamic outdoor or competitive golf scenarios.

Results and Evaluation

This section presents the testing methodology, performance outcomes and final system evaluation of ParSight. We begin by detailing the structured incremental testing pipeline used to validate individual subsystems before full-system integration, Verification results are then benchmarked against key performance metrics to assess the functionality and responsiveness of the prototype. Furthermore, to assess real-world utility, we conducted validation trials which aim to recreate dynamic golf scenarios, evaluating how well ParSight delivers on its intended purpose. Finally, we reflect on system limitations and outline opportunities for future improvements to enhance performance, robustness, and deployability in outdoor golfing environments.

Testing Pipeline Strategy

To ensure the reliability, safety, and effectiveness of ParSight, we employed a structured, incremental testing pipeline that prioritized subsystem verification before full-system integration. This allowed us to catch potential failures early, tune critical parameters iteratively, and gradually build up to our most complex demonstrations involving autonomous flight and fast-moving targets. Each stage was designed to isolate a specific interaction, be it detection, control, or actuation, so that issues could be addressed without cascading complexity.

Incremental Testing Steps

Our testing strategy began with non-flight validation using laptop-based detection. Once the HSV colour filtering and contour selection pipeline proved robust offline, we transitioned to static onboard testing to verify that real-time image acquisition, ROS topic synchronization, and visualization tools (like RViz) worked as intended. Following this, we introduced actuation logic through simulated motion by manually moving the drone and analyzing how tracking setpoints responded. With confidence in both perception and communication, we introduced autonomous flight for the first time in the "slow hover" tests, where the drone visually locked onto a ball without initiating movement. Subsequent tests introduced incremental motion control, beginning with fixed 1 cm steps, progressing to proportional-only control, and finally stabilizing on a tuned PD controller. These tests helped us measure how precisely the drone could follow the ball, how quickly it could correct errors, and how smooth its response was under different motion profiles. Our final two test stages pushed the limits of system responsiveness. Using an RC car and a mini catapult to simulate fast motion and ball launches, we validated whether ParSight could track a ball through acceleration, bounce, and roll, then hover reliably above its final location. Each test in this pipeline was chosen not just for complexity, but for the specific insights it would offer into detection latency, control response time, and overall robustness.

A full summary of all tests, including test type, control approach, and flight status, is shown in *Table 3* below, illustrating the progression from ground-based detection to our final integrated MVP demonstration. The gradual evolution of testing complexity enabled us to debug safely, tune effectively, and build confidence that each part of ParSight was working before deploying the next.

Table 3. *Overview of ParSight's testing pipeline, showing test types, control strategies, and flight status.*

Test Name	**Test Type**	**Control Type**	**In Flight?**
Laptop Detection Test	Detection	-	-
Drone Detection Test	Detection	-	NO
Drone Tracking Test	Tracking	Fixed Step (1 cm)	NO
Drone Detection Slow Movement (S)	Detection	-	YES
Drone Tracking Slow Movement (1cm)	Tracking	Fixed Step (1 cm)	YES
Drone Tracking Slow Movement (P)	Tracking	P Control	YES
Drone Tracking Slow Movement (PD)	Tracking	PD Control	YES
Drone Tracking Fast Movement (PD)	Tracking	PD Control	YES
MVP: Drone Tracking Catapult Launch	Tracking	PD Control	YES

Verification: Metric-Based System Performance

Verification was conducted by comparing system performance against the defined MVP metrics, as outlined in *Table 4*. This allowed us to assess whether the core subsystems of ParSight functioned correctly and in accordance with the design requirements. All perception-related requirements were met or exceeded. The RGB camera consistently delivered a frame rate of 30 Hz, ensuring real-time image acquisition. The object detection pipeline achieved a true positive rate (TPR) of 95% with a false positive rate (FPR) of only 3%, indicating robust and accurate detection even in cluttered environments and varying lighting conditions. The system maintained a detection rate of 30 Hz with no dropped frames on the Jetson Nano, confirming end-to-end real-time capability. Control-side metrics showed mixed results. The flight control planning speed was measured at 14.1 ms, well under the 20 ms target. However, the reaction speed of 350 ms and tracking precision of 44.2 pixels fell short of the desired thresholds of 150 ms and 40 pixels, respectively. These shortcomings were largely attributed to hardware constraints, specifically the computational limitations of the Jetson Nano, ROS message-passing latency, and the smoothing effects of our PD controller which traded responsiveness for stability. Despite these limitations, the drone consistently responded to motion cues and adjusted its position effectively during all relevant test scenarios, including both our slow-tracking trials and the final MVP demonstration. Processing latency was a key verification metric, and ParSight maintained a steady 30.5 ms, well below the 100 ms limit. This demonstrated that all sensing, detection, and actuation commands could be completed within a single control loop cycle, reinforcing the system's readiness for real-time use. In total, 8 out of 10 system metrics were successfully met, and the remaining two were close to target. The system exhibited consistent performance and safety throughout flight trials, verifying that the design was both reliable and technically sound.

Table 4. *Summary of ParSight's system performance against defined MVP performance requirements, highlighting achieved and unmet benchmarks.*

Code/Name	**MVP Metric**	**MVP Result**
IMG-P-001: Camera Data Rate	**> 20 Hz**	30 Hz
OBD-P-001: Object Detection Rate	**> 20 Hz**	30 Hz
OBD-P-002: Object Detection TPR	**> 90%**	95%
OBD-P-003: Object Detection FPR	**< 5%**	3%
OBD-P-004: Object Detection Robustness	**> 85%**	95%
FLC-P-001: Flight Control Planning Speed	**< 20 ms**	14.1 ms
FLC-P-002: Flight Control Reaction Speed	**< 150 ms**	**350 ms**
FLC-P-003: Flight Control Tracking Precision	**< 40 pixels**	**44.2 pixels**
FLC-P-004: Flight Control Speed	**TRUE**	TRUE
SLR-P-001: Drone Processing Latency	**< 100 ms**	30.5 ms

Validation: Real-World Tracking Capability
Validation focused on whether ParSight fulfilled its intended purpose, serving as a real-time, autonomous visual aid to help golfers locate their ball after a shot. This was evaluated through a series of progressively more complex flight tests, culminating in a full-system MVP demonstration involving a high-speed catapult launch. In this final test, ParSight successfully detected the golf ball, maintained lock during the ball's flight, and autonomously repositioned itself to hover over the final resting location. The drone remained within 6 cm of the ball's final position, visually marking its location without requiring user input. The entire process was fully autonomous and repeatable, demonstrating that ParSight could perform its function in dynamic, unstructured environments similar to those encountered on a golf course. The successful execution of this test validated that the system met its overarching user-facing objective. While reaction time and tracking precision were slightly below target, the real-world utility of the system was not compromised. The drone's visible presence above the ball significantly reduced search time and enabled autonomous tracking without modifying the core golfing experience. As such, ParSight can be considered a validated solution for improving golf ball visibility, particularly for older players facing age-related vision challenges.

Future Work
While ParSight achieved its core design objectives, several areas were identified for further development. One key limitation was the tracking latency and precision, constrained by the compute capacity of the Jetson Nano and ROS communication overhead. Upgrading to a more capable platform, such as the Jetson Orin Nano or an edge device with GPU acceleration, would allow for tighter control loops and smoother real-time responses. In terms of perception, our final implementation relied on HSV-based colour filtering due to its speed and simplicity. While this method enabled fast, lightweight segmentation, it remained sensitive to environmental noise such as red floor markings and lighting changes. One future direction is returning to our earlier YOLO-based segmentation pipeline, which, despite initial performance bottlenecks, offered improved robustness. With upgraded hardware, YOLOv8 or other CNN-based models (ie. MobileNet or EfficientDet) could be run in real time using TensorRT optimization, enabling stronger generalization across ball appearances, surfaces, and lighting conditions. Beyond perception, predictive motion planning could further enhance tracking. Currently, the drone responds reactively to the ball's position. Incorporating lightweight trajectory estimation or Kalman filtering could help anticipate motion and improve convergence under faster ball movement. We also envision adding dynamic altitude control, enabling the drone to adjust height based on ball speed or vertical arc to improve visibility and mimic a more natural "follow" behaviour. Real-world field deployment, on a driving range or course, remains a critical next step to validate the system's practical effectiveness with senior golfers in outdoor environments. These improvements would help transform ParSight from a proof-of-concept into a deployable assistive drone, offering real-time ball tracking in a wide range of real-world golf scenarios.

Lessons Learned and Team Reflection

This section concludes with a summary of the challenges encountered throughout the design process, both technical and collaboration-related. From these challenges, we have also reflected on things we would do differently had we completed the design again and provided advice for future teams.

Challenges

Throughout the project, we encountered several challenges that impacted our progress. Hardware errors were a significant issue, large crashes would result in seemingly stochastic errors due to the identifiable source of the problem. For example, the RealSense camera stopped working, which caused pose estimation errors, and damages to the Cube Orange+ flight controller led to discrepancies between vision pose and setpoint positions that could not be resolved. Limited flight time and battery life also posed constraints; course restrictions on available flight times and the batteries provided made it difficult to make necessary adjustments to the colour detection algorithms and the PD controller. Additionally, knowledge sharing within the team was limited. We did not take the time to teach each other about the components we were individually working on, such as how to adjust the colour detection algorithm, tune the PD controller, launch service calls, or fly the drone. As a result, all four members mostly had to be present for any progress to be made. Finally, decision-making was often slow, as all four members needed to agree before proceeding. This was particularly challenging during our limited flight sessions, where quick decisions were crucial to addressing issues with the drone in real time.

Areas for Improvement

There are several areas where our team could improve in future projects. First, we should prioritize simpler solutions early on. We pivoted from YOLO segmentation to HSV colour detection after realizing YOLO was too computationally intensive. In hindsight, starting with HSV filtering would have been more effective, as it was lightweight, simple to implement, and could have served as a proof of concept before progressing to more complex algorithms. Additionally, although our incremental testing plan helped improve the safety and success of our design, we never tested the algorithm in simulation. Recreating the environment and drone in Gazebo would have allowed us to verify that the system's behaviour met expectations before deployment. Building on this, automated testing, such as unit tests, could have been introduced to ensure that new code pushes didn't break critical drone functions. For instance, flight tests in simulation could have been triggered automatically with each push to the GitHub repository. Lastly, better documentation would have helped address the knowledge-sharing challenge we faced. By documenting our code throughout the design process, all team members could have better understood each component and contributed more effectively.

Advice for Future Teams

For future teams, there are several key practices that can help streamline development and avoid common pitfalls. First, always maintain fallback code that is known to work and remains unchanged, this allows for quick sanity checks to determine whether issues stem from new code changes or underlying hardware problems. It's also crucial to ensure that your development environment is properly set up, well-documented, and consistently tracked. This prevents time from being wasted on setup issues and helps onboard new team members more efficiently. Taking the time to go through ROS tutorials can also be incredibly helpful for understanding how workspaces and packages should be structured, as well as learning how to use Git effectively. Finally, don't overlook DevOps practices, automating deployment processes and building debugging tools like a GUI may seem secondary, but they are invaluable for troubleshooting and maintaining system reliability in the long run. Investing time in these areas early on can significantly improve project outcomes.

References

[1] “A New Age in Golf? - National Golf Foundation.” Accessed: Apr. 21, 2025. [Online]. Available: https://www.ngf.org/short-game/a-new-age-in-golf/

[2] N. Erdinest, N. London, I. Lavy, Y. Morad, and N. Levinger, “Vision through Healthy Aging Eyes,” *Vision*, vol. 5, no. 4, p. 46, Sep. 2021, doi: 10.3390/vision5040046.

[3] “How Golf Becomes More Difficult When You Get Older,” Golfshake.com. Accessed: Feb. 10, 2025. [Online]. Available:http://www.golfshake.com/news/view/20587/How_Golf_Becomes_More_Difficult_When_You_Get_Older.html

[4] “Does anyone have issues with eyesight and tracking their golf ball?,” GolfWRX. Accessed: Feb. 10, 2025. [Online]. Available: https://forums.golfwrx.com/topic/679066-does-anyone-have-issues-with-eyesight-and-tracking-their-golf-ball/

[5] “Shot Tracer,” Shot Tracer. Accessed: Feb. 10, 2025. [Online]. Available: https://www.shottracerapp.com

[6] S. V. G. Contributor, “PGA TOUR Brings Back Live Drone Tracer Tech for FedExCup Playoffs and Presidents Cup,” Sports Video Group. Accessed: Feb. 10, 2025. [Online]. Available: https://www.sportsvideo.org/2024/08/19/pga-tour-brings-back-live-drone-tracer-tech-for-fedexcup-playoffs-and-presidents-cup/

[7] “These insanely fast racing drones capture golf shots as you’ve never seen them,” Golf. Accessed: Feb. 10, 2025. [Online]. Available: https://golf.com/news/racing-drone-golf-shots/

[8] “Drones Follow the Ball on PGA Golf Tour.” Accessed: Feb. 10, 2025. [Online]. Available: https://www.thedroningcompany.com/blog/www.thedroningcompany.com/blog/drones-follow-the-ball-on-pga-golf-tour

[9] S. French, “The best follow-me drone for tracking sports like skateboarding, biking,” The Drone Girl. Accessed: Feb. 10, 2025. [Online]. Available: https://www.thedronegirl.com/2020/12/09/best-follow-me-drone/

[10] “Color Detection.” Accessed: Apr. 21, 2025. [Online]. Available: https://www.mathworks.com/help/simulink/supportpkg/android_ref/color-detection.html

[11] “Color Sensing with Computer Vision,” Roboflow Blog. Accessed: Apr. 21, 2025. [Online]. Available: https://blog.roboflow.com/color-sensing-with-computer-vision/

[12] “Color Vision: Is it Better for Robotics?” Accessed: Apr. 21, 2025. [Online]. Available: https://blog.robotiq.com/color-vision-is-it-better-for-robotics

[13] S. Hutchinson, G. D. Hager, and P. I. Corke, “A tutorial on visual servo control,” *IEEE Trans. Robot. Autom.*, vol. 12, no. 5, pp. 651–670, Oct. 1996, doi: 10.1109/70.538972.

[14] “An Overview of Visual Servoing for Robot Manipulators - Technical Articles.” Accessed: Apr. 21, 2025. [Online]. Available: https://control.com/technical-articles/an-overview-of-visual-servoing-for-robot-manipulators/

[15] “The PID Controller & Theory Explained.” Accessed: Apr. 21, 2025. [Online]. Available: https://www.ni.com/en/shop/labview/pid-theory-explained.html

[16] “Understanding PID Controller: Types, Working, and Applications.” Accessed: Apr. 21, 2025. [Online]. Available: https://multispanindia.com/blog-detail.php/pid-controller-types-working-and-its-application

[17] Y. Hao *et al.*, “Understanding the Impact of Image Quality and Distance of Objects to Object Detection Performance,” Sep. 17, 2022, *arXiv*: arXiv:2209.08237. doi: 10.48550/arXiv.2209.08237.

[18] M. Breton, “Defence Research and Development Canada Recherche et développement pour la dé fense Canada Overview of two performance metrics for object detection algorithms evaluation Defence Research and Development Canada Reference Document CAN UNCLASSIFIED”, [Online]. Available: https://cradpdf.drdc-rddc.gc.ca/PDFS/unc342/p811371_A1b.pdf

[19] R. Ferrer-Urbina, A. Pardo, W. A. Arrindell, and G. Puddu-Gallardo, “Comparison of false positive and false negative rates of two indices of individual reliable change: Jacobson-Truax and Hageman-Arrindell methods,” *Front. Psychol.*, vol. 14, Jul. 2023, doi: 10.3389/fpsyg.2023.1132128.

[20] “NexiGo 1080P Business Webcam with Software, Dual Microphone & Privacy Cover, NexiGo N660 USB FHD Web Computer Camera, Plug and Play, for Zoom/Skype/Teams/Webex, Laptop MAC PC Desktop : Amazon.ca: Electronics.” Accessed: Apr. 21, 2025. [Online]. Available: https://www.amazon.ca/Upgraded-Microphone-NexiGo-Reduction-Teaching/dp/B08BHX7GYY?th=1

[21] “Jetson Nano vs Jetson Orin Nano 8 GB.” Accessed: Apr. 21, 2025. [Online]. Available: https://technical.city/en/video/Jetson-Orin-Nano-8-GB-vs-Jetson-Nano