Controls & Vision: Turkey Deterrent Robot Final Report

Section 005 – Professor Awtar
ME 450 Team 17
Brady Buten
Eric Giles
Yuxi (Ethan) Chen
Zachary Sun

EXECUTIVE SUMMARY

Michigan's Department of Natural Resources (DNR) has been receiving increasing reports of wild turkey related disturbances across urban Michigan, including on the University of Michigan North Campus. These incidents include damage to property, harassment of pedestrians, and the obstruction of walkways and building entrances. Currently, the university faculties and maintenance is limited to staff physically chasing birds, which is operationally unsustainable and fails to deter the birds long term. Sponsored by Jared Duquette of the DNR, our goal was to develop an autonomous canine robot that humanely hazes wild turkeys, adjusting their movement behaviors away from key locations over time.

The project was divided into three subteams: Team 17 (Controls & Vision), Team 18 (Drivetrain), and Team 19 (Canine Design). Our team (Team 17) was responsible for the system's intelligence, encompassing computer vision, autonomous navigation and controls. This required constant coordination between subteams, culminating in a two week integration phase in April at the GGBL X50 lab to assemble the final product. Team 17 followed three primary objectives: achieving reliable object detection, implementing a navigation system to avoid obstacles and maintain zone boundaries, and integrating high level controls with the mechanical hardware from Teams 18 and 19.

Our team delivered an autonomous canine robot prototype capable of deterring turkeys from a designated zone on U-M's North Campus. The prototype features a wheeled drivetrain, an animatronic canine, and a real time vision system utilizing a custom trained YOLO object detection model to identify turkeys, humans, and vehicles. For navigation, the system employs an A* path planning algorithm to compute optimal routes and a Model Predictive Control (MPC) trajectory tracker to generate feasible motion commands for the drivetrain. We also developed a wireless user interface that provides live video feeds, geolocation of the device, and manual control of onboard actuators. The prototype was validated through field testing with a decoy and live turkeys, where the robot successfully induced a flight response.

This project serves as a foundational proof of concept for subsequent project iterations and potential municipal deployment. To reach full operational capacity, future steps should include incorporating 3D LiDAR and Simultaneous Localization and Mapping (SLAM) for real time environment mapping, which would replace the current need for pre mapped static obstacles. Furthermore, migrating the software architecture to the ROS2 framework will provide a more robust platform for expanded autonomy functions. To support long term, independent operation, we also recommend the development of automatic charging stations and the transition to custom PCB electronics and integrated wire harnessing.

TABLE OF CONTENTS

ABSTRACT **7**
1 PROJECT INTRODUCTION AND BACKGROUND **7**
1.1 Motivation 7
1.2 Behavioral Context 9
1.3 Benchmarking 10
1.4 Sponsorship 11
2 SUBTEAM STRUCTURE **11**
2.1 Controls and Vision - Team 17 13
2.2 Chassis and Suspension - Team 18 13
2.3 Canine Design - Team 19 13
3 ZONE DEFINITION **13**
4 OBJECTIVE, SCOPE AND DELIVERABLES **15**
4.1 Controls and Vision Subteam Scope 15
4.1.1 Computer Vision 15
4.1.2 Navigation 16
4.1.3 Control Actuation 16
4.1.4 User Interface 16
5 PROJECT PLAN **16**
6 DESIGN PROCESS **18**
6.1 Information Sources 18
6.2 Engineering Standards 19
7 DESIGN CONTEXT **19**
7.1 Intellectual property 20
7.2 Sustainability 20
7.3 Ethical Considerations 21
8 USER REQUIREMENTS AND SPECIFICATIONS **21**
8.1 Environment Detection Requirements 24
8.1.1 Accuracy 24
8.1.2 Speed 25
8.2 Movement Requirements 25
8.2.1 Safety 26
8.2.2 Turkey Deterrence 26
8.3 User Functions Requirements 27
9 CONCEPT GENERATION AND SELECTION **27**
9.1 System Navigation Options 27
9.2 Camera Mounting 29

10 HARDWARE & SOFTWARE SELECTION.....31
10.1 Computer Vision Model Selection.....31
10.2 Camera Selection.....33
10.3 Navigation Selection.....34
11 PRELIMINARY DESIGN CONCEPT.....35
11.1 Scanning Protocol.....36
11.2 Chase Protocol.....37
11.3 Alert Protocol.....37
11.5 User Interface.....39
12.2 State Planner.....41
12.3 Computer Vision.....43
12.3.1 Computer Vision Model.....43
Camera Detection Algorithm.....43
12.3.2 Distance Estimation Algorithm.....44
12.4.1 Zone Mapping.....46
12.4.2 Zone and Obstacle Representation.....47
12.4.3 Path Generation.....47
12.4.4 Trajectory Tracking.....48
12.5.1 Motor Control.....49
12.5.2 PID Gain Tuning.....50
12.5.3 Firmware Safety Routines.....50
12.6 Localization.....51
12.6.1 Predict Step.....51
12.6.2 Update Step.....52
12.7 Canine Actuation.....52
12.7.1 Servo Control.....53
12.7.2 Audio.....54
12.9 Mechanical Assemblies.....56
13.1 Computer Vision.....59
13.1.1 Model-Level Evaluation.....59
13.1.2 System-Level Evaluation.....60
13.1.3 Field Testing.....61
14. PROBLEM ANALYSIS AND ITERATION.....62
14.1 Computer Vision.....62
14.1.1 High-Resolution Model Performance Degradation and Dataset Scaling.....62
14.1.2 Pose Sensitivity in Distance Estimation and Data-Driven Improvement.....62
14.2 Controls.....63
14.2.1 Control Actuation.....63

14.2.2 Navigation....63
15 PROBLEM DOMAIN AND ANALYSIS....64
15.1 Computer Vision....64
15.2 Controls....65
17 PERFORMANCE DEMONSTRATION....72
18 DISCUSSION....75
18.1 Subteam Scoping and Problem Definition....75
18.2 Design Critique....76
18.2.1 Computer Vision....76
18.2.2 Navigation....77
18.2.3 Controls....78
18.2.4 System Integration....79
18.3 Project Risks....80
19 RECOMMENDATIONS....81
19.1 Expanded Autonomy Functions....81
19.2 Vision Hardware and Online Obstacle Detection....81
19.3 GPS Tracking....81
19.4 ROS2 Framework....82
19.5 Power Stations for Automatic Charging....82
19.6 Wire Professionalism and Ease of User Operation....82
20 REFLECTION....83
21 CONCLUSION....85
ACKNOWLEDGEMENTS....86
REFERENCES....87
TEAM BIOS....89
APPENDIX....91
A.1 SOFTWARE FILES AND DOCUMENTATION....91
A2 BOM SECTION....91
A3 CANINE ACTUATOR SPECIFICATIONS....92
A3.1 Actuator Design Scope....92
A3.2 Audio Commands....92
Maximum Servo Angles and Speeds Commands....92
A3.3 Fixed Motor Commands....93
IdlePose....93
AlertPose....93
AggressivePose....93
A3.4 Oscillating Motor Commands....93
WalkGait....93

ExcitedGait..... 94
AlertStep..... 94
HeadScan..... 95
A3.5 Chase Protocol..... 95
Protocol 1 – Aggressive Pursuit..... 95
Protocol 2 – Burst Attack..... 95
Protocol 3 – Growl and Chase..... 95
A3.6 Alert Protocol..... 96
Protocol 1 – Watchful Dog..... 96
Protocol 2 – Territorial Warning..... 96
Protocol 3 – Short Warning Burst..... 96
A3.7 Scan Protocol..... 96
Mode 1 - Traveling between waypoints..... 96
Mode 2 - Stationary Scan..... 96
Mode 3 - Stationary Crouch..... 97
A.4 SOFTWARE PROTOCOL..... 97
1A: VisionObject Protocol..... 98
1B: VideoFeed Protocol..... 98
2A: Canine Protocol..... 98
2B: ServoFeedback Protocol..... 99
3A: DriveCommand Protocol..... 99
3B: SlipFeedback Protocol..... 99
4: RobotPose Protocol..... 99
5: MacroState Protocol..... 100
6: MotionCommand Protocol..... 101
7A: UWB Protocol..... 102
7B: Accelerometer Protocol..... 102

ABSTRACT
Rising wild turkey populations in urban Michigan have led to increasing pedestrian conflicts and vehicle damage. This project presents an autonomous robotic canine designed to deter turkeys from designated exclusion zones through hazing behaviors, encouraging relocation to more suitable wilderness areas. The system employs computer vision for turkey detection and identification, an A* path planning algorithm, and a Model Predictive Control (MPC) trajectory tracker. Custom control software manages canine locomotion and drivetrain actuation. The result is a minimally intrusive, automated wildlife management solution suitable for municipal deployment.

1 PROJECT INTRODUCTION AND BACKGROUND

Turkey attacks have been increasing in urban areas of Michigan, particularly during the spring mating season when turkeys become highly territorial. During this time, reports of intimidation and attacks on pedestrians have risen substantially, causing obstruction of building entrances, sidewalks, and parking lots, and creating a growing burden on maintenance and wildlife control services. Sponsored by the Department of Natural Resources, our project scope aims to develop an autonomous canine capable of guarding specified high-nuisance areas on the University of Michigan North Campus. By presenting a consistent predator-like presence, the device will alter turkey movement patterns, encouraging them to relocate to more wilderness habitats and away from high-traffic pedestrian locations.

1.1 Motivation

The project is motivated by a need to restore safety and operational efficiency to the urban areas in Michigan, and more specifically the University of Michigan North Campus. Data from a student survey sent out in winter 2026 indicates a majority of respondents have felt unsafe around turkeys before, as shown below in Figure 1 [1].

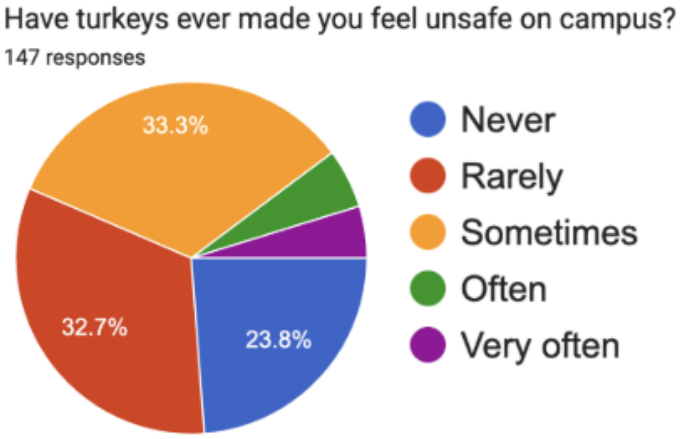


Figure 1. Campus poll investigating feelings of safety around turkeys

Sightings often involve aggressive behavior by turkeys, which creates a safety hazard for students and a burden on maintenance, such as the university grounds crews who are responsible for property damage. The university needs a solution to prevent turkeys from entering high traffic areas. A successful robot would deter the birds in a humane way, leading to a reduction in staff labor and frequency of negative human-wildlife interactions. Figure 2 shows an example of turkeys disrupting campus operations.



Figure 2. Image of two turkeys blocking a University bus on North Campus

The DNR has considered several solutions to combat the increased presence of turkeys in urban areas of Michigan. While extermination is feasible, the university and locals have expressed strong opposition to this approach, due to ethical considerations as well as the potential of disturbing residents and students. Another commonly used method is translocation, or the intentional relocation of animals into a different environment. Although possible, this process can be difficult and expensive, and also carries the risk of spreading diseases, especially to areas where the wildlife may have lower resistances to such diseases. The unpredictable behavior of animals after translocation also poses an issue. According to the Global Re-introduction

Perspective Series, conducted by the International Union for Conservation of Nature (IUCN), a survey of 293 species translocation efforts concluded that the greatest challenge was changes in animal behavior in their new environments [2]. Due to the unpredictability of this approach, the DNR has requested that we utilize hazing instead, to gradually shift the movements of turkeys away from key areas, over time.

1.2 Behavioral Context

The presence of these birds on campus can be explained by several ecological factors. The human shield hypothesis suggests that wild turkeys intentionally enter areas of high activity because these locations act as a refuge, shielding them from natural predators like coyotes who avoid densely populated zones. Research conducted in Georgia indicates that female turkeys often stay closer to human activity during high risk nesting stages, specifically incubation and brooding, to enhance their survival and reproductive success [3]. As human encounters are typically nonlethal, turkeys likely learn humans are a lower threat than natural predators.

Risk assessment in birds is commonly measured by Flight Initiation Distance (FID), the distance at which an individual flees from an approaching potential predator. Urban dwelling birds often exhibit a reduced FID [4], becoming so habituated to human presence that they no longer flee from passive threats. This lack of fear combined with a shortage of natural predation leads to high density colonization.

Wild turkeys are omnivorous opportunistic ground feeders, whose foraging habits often lead to agricultural damage. They use their strong legs to tear up coverage in search of insects and seeds. This results in the destruction of gardens, flower beds, and other landscaping. In vineyards, turkeys damage crops by plucking large sections of berries from grape clusters [5]. The birds are also territorial. Turkeys frequently perceive their own reflections in windows and shiny vehicle surfaces as an intruder, and will begin aggressively pecking and scratching those surfaces.

Traditional stationary deterrents fail because turkeys have the cognitive ability to adapt quickly to predictable non threatening stimuli. Bioacoustic techniques, such as broadcasting alarms and distress calls have been ineffective in long term trials [6]. The birds begin to ignore the sounds once they realize no physical threat is present. Scaring devices like propane cannons, Mylar tape, and effigies also only work short term. Turkeys rapidly habituate to them if the locations and frequency are predictable. Thus, effective management requires active and unpredictable hazing techniques. We therefore conclude that a mobile robot solution is required to provoke a flight response, and deter turkeys from key areas.

1.3 Benchmarking

In order to develop a better understanding of our solution's performance, feasibility, cost and reliability, our team utilized benchmarking in order to evaluate existing solutions, and establish realistic performance targets.

One such product that we found to be particularly relevant is Bird Control Group's AVIX Autonomic Mark II, seen in Figure 3 below [7]. This solution uses computer vision detection to identify birds, and autonomously point the laser in the target's direction to threaten them away.



Figure 3. Photograph of Bird Control Group's AVIX Autonomic Mark II solution.

This product can provide us key insights into detecting biological targets (birds), operating in outdoor environments, and tracking moving objects with high precision. This product has been implemented in various contexts since 2014, and has resulted in a greater than 70% reduction in bird numbers over the course of that period [7]. This gives us confidence that we can use Bird Control Group's framework to benchmark detection latency targets, false positive mitigation strategies, and operation autonomously in the outdoors. Unfortunately, as our sponsor mentioned, turkeys do not share the same fear response to lasers as smaller aerial birds, and therefore our solution will have to rely on a more complex canine biomimicry system, in order to invoke the same response.

Another example we looked at was Bird-B-Gone's motion activated scarecrow sprinkler, seen in Figure 4. below [8]. Designed for deterring large birds away from protected areas, the autonomous system functions through spraying streams of water in response to triggered motion sensors.



Figure 4. Photograph of Bird-B-Gone's Motion Activated Bird Scarecrow Sprinkler.

Although this product uses infrared detection rather than full CV, it allows us to visualize and evaluate the detection-to-deterrent pipeline, which is central to our canine robot. From this solution, we are able to benchmark detection range, trigger latency, outdoor robustness, and power consumption. The issue with this particular solution is that it relies on PIR, and therefore is unable to distinguish between turkeys and other animals, or even pedestrians. Since we are trying to clear out zones that are regularly traversed by pedestrians, this is not a valid solution, and we will need to rely on computer vision in order to make that distinction.

1.4 Sponsorship

This project is sponsored by the Michigan Department of Natural Resources (MDNR) under the guidance of Dr. Jared Duquette, a Human-Wildlife Interactions Specialist. Through conservation efforts, the MDNR has restored the statewide wild turkey population to approximately 200,000 birds [9]. This population rebound has caused turkeys to move into increasingly suburban areas. Dr Duquette has generously offered the teams a total budget of $10,000 to create an autonomous canine decoy. Our teams initially planned on agnostic requirements for the deterrent design, however Dr. Duquette ensured that mimicking the sounds and movements of dogs is most effective, as they are natural predators to the birds, who have keen eyesight and hearing. The sponsor has also requested the device be capable of operating in varying terrain and weather conditions, which will be experienced on the university campus.

2 SUBTEAM STRUCTURE

Our project structure is unique within ME 450, as three dedicated teams are collaborating on a single deliverable. To define each team's area of focus, we developed a high-level system concept to partition responsibilities across subteams. As shown in Figure 5, the system consists of a single autonomous mobile robotic drivetrain with a realistic animatronic canine mounted above it.

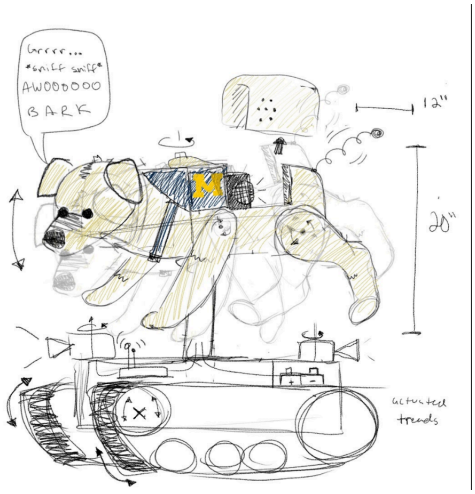


Figure 5. Preliminary system concept. The animatronic canine is mounted atop the motorized drivetrain, with high-level autonomy implicitly defined.

The subteam structure is organized as follows: Team 17 (Controls & Vision) is responsible for sensing, autonomy, and system control; Team 18 (Chassis & Suspension) focuses on the powertrain and drivetrain assembly; and Team 19 (Canine Design) leads the mechanical design, aesthetics, and behavioral features of the animatronic canine, including its deterrence characteristics.

The elements presented in the high-level system concept are intended to differentiate subteam scope, while detailed design decisions remain within the respective subteam domains. The key responsibilities for each subteam is depicted below in Figure 6.

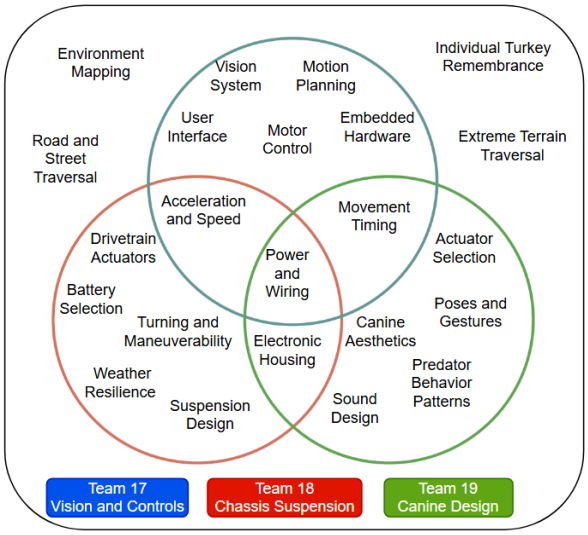


Figure 6. A high-level overview of subteam scope and shared responsibilities. Team 17 is outlined in the blue uppermost circle, Team 18 in the red bottom-left circle, and Team 19 in the green bottom-right circle. Elements outside the project scope are not enclosed by any circle.

2.1 Controls and Vision - Team 17

Team 17 is responsible for the programming and embedded hardware components of the project. This includes developing a real-time vision system capable of classifying turkeys, humans, and vehicles within the predefined zone, as well as selecting camera and computational hardware to ensure accurate and responsive detection. Team 17 will also create the user interface, supporting manual inputs and providing real-time visual and system feedback. Additionally, the subteam handles overall navigation, motion planning, low-level control drivers, and the selection of embedded hardware for motor control and encoder feedback.

2.2 Chassis and Suspension - Team 18

Team 18 is responsible for designing the drivetrain and ensuring maneuverability across uneven outdoor terrain. This includes selecting wheels, motors, and suspension components to support reliable movement within the zone, as well as defining operational parameters such as acceleration, turning radius, and speed. Team 18 also provides mechanical housing and protection for embedded hardware and non-motorized actuators, including cameras, power amplifiers, and speaker systems. Additionally, Team 18 manages battery selection and power distribution for all subteams.

2.3 Canine Design - Team 19

Team 19 is responsible for the mechanical and behavioral design that creates the “fear factor” to deter turkeys. This includes building a canine-like structure and selecting actuators to support motorized gestures and poses, while ensuring mechanical safety by addressing pinch points and rough edges. The subteam also handles sound design, including barking, growling, and other predator-like sounds.Additionally, Team 19 guides Teams 17 and 18 on implementing predator behaviors to maximize the system’s effectiveness in deterring turkeys.

3 ZONE DEFINITION

Per sponsor intent, the envisioned end-state product would be deployed to municipalities and operate over a broad area with substantial roaming capability. However, given project timeline constraints, the scope of this deliverable is limited to operation within a single high-impact zone. The zone is directly East of the UoM Climate and Space Building, shown on a map below in Figure 7.



Figure 7. A map snippet of U-M North Campus with the defined operational zone outlined in red. The zone is bounded by the sidewalk to NC28 to the north, the Climate and Space Building to the east, Hayward Street to the south, and the wooded area to the west.

To select this zone, we obtained a map of turkey incident reports by location from our UM Maintenance contact, Shaun Ashley. This dataset is shown in relation to our selected operational zone in Figure 8.

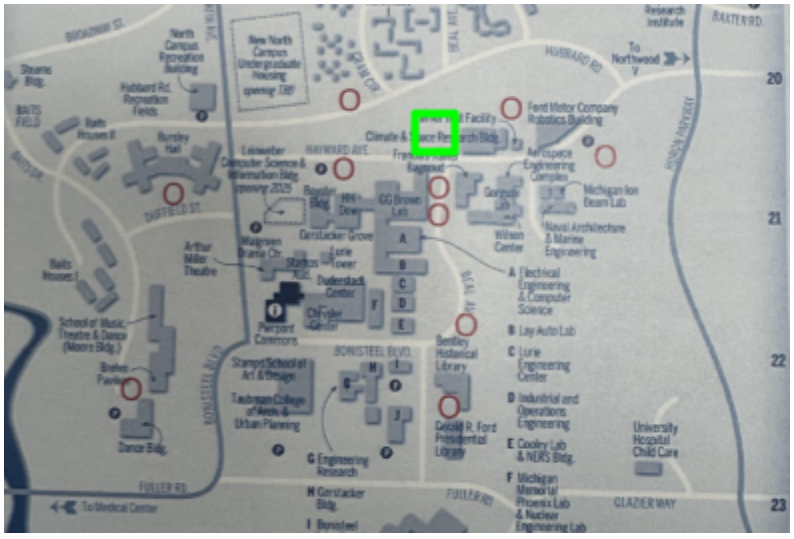


Figure 8. Map of turkey incident reports provided by UM Maintenance. Locations outlined in red indicate reported turkey calls during the 2025 calendar year. The green square denotes the annotated boundaries of the selected operational zone.

In selecting this zone, we balanced testing feasibility with projected turkey activity. The area is centrally located within the incident data provided by UM Maintenance. In addition, anecdotal observations and recorded video evidence indicate recurring turkey presence and human–turkey conflict within the defined boundaries. The terrain is relatively level, with minimal foliage and no significant physical hazards, making it suitable for controlled testing and repeatable operation. The area also experiences comparatively high pedestrian traffic relative to similar nearby areas, increasing the relevance and potential impact of our deliverable in this zone.

4 OBJECTIVE, SCOPE AND DELIVERABLES

By the end of May 2026, the three subteams will jointly deliver an autonomous canine robot designed to deter turkeys from a single high-impact zone. The system will consist of a wheeled robotic drivetrain with an animatronic canine mounted above it. The software components developed by our subteam are detailed in a subsequent section. Project success will be defined by the system's ability to reduce turkey entry into the designated zone and serve as a foundational proof of concept for subsequent project iterations.

Given project timeline constraints, the final deliverable will operate within the defined zone for a limited duration of several hours and will require substantial manual intervention to function. The intended long-term vision is a fully autonomous system capable of covering a significantly larger municipal area. Terrain capability in the current prototype will be restricted to dirt, grass, and pavement, whereas a deployable version would need to accommodate more demanding conditions such as snow, deep mud, and heavy rain. The design is not intended for immediate handoff to maintenance personnel, as additional iterations would be necessary to address usability, robustness, and deployment considerations.

4.1 Controls and Vision Subteam Scope

Within the Controls and Vision subteam, our technical stack is organized around three primary pillars: Computer Vision, Navigation, and Control Actuation. To enhance operational safety, a remote user interface will also be included to interact with these pillars. The scope and responsibilities of each pillar, along with the UI, are outlined below.

4.1.1 Computer Vision

The computer vision system will be responsible for accurately detecting and estimating the locations of turkeys, pedestrians, and vehicles within the operational zone. It must operate reliably under varied conditions, including different lighting levels, angled or partially obstructed views, and motion blur. Our approach involves constructing and training a custom computer vision model using both existing datasets and newly acquired camera data. Additionally, the subteam will select and mount the camera hardware, evaluating available options and, if necessary, designing a custom mechanical mount to optimize zone coverage.

4.1.2 Navigation

The navigation system will be responsible for generating feasible paths based on the robot's initial pose and target destinations within the zone. It will utilize vision data to estimate the positions of turkeys and humans, enabling dynamic path planning that avoids collisions and prevents the robot from leaving the defined boundaries. Our subteam will develop the software architecture to support real-time updates, allowing the system to respond to the dynamic movements of both humans and turkeys while maintaining safe and efficient navigation

4.1.3 Control Actuation

Once a feasible path is generated, the control actuation system will convert it into executable control inputs. This includes developing low-level hardware drivers to interface with and control motor drivers for the drivetrain, in coordination with other subteams. Our subteam will also handle the selection of motor drivers and other electronic components, excluding the motors themselves. Additionally, we will estimate the robot's pose within the zone, employing state estimation techniques to maintain accurate real-time tracking of the system's position and orientation.

4.1.4 User Interface

Safety is a primary concern for our project deliverable. To prevent the system from leaving the designated zone or mistakenly pursuing a human, a remotely actuated interface is critical. The user interface will also enhance testability and simplify system integration during development. It will provide mechanisms for manual motor control and an emergency stop, as well as a display showing the predicted robot pose and vision-based object classifications to support debugging and real-time monitoring.

5 PROJECT PLAN

Shown below in Table 1 is a project timeline designed to align ME 450 course deliverables with the development of our control and vision subsystems. The ME 450 presentations serve as natural checkpoints with Teams 18 and 19, enabling comparison of high-level design decisions during DR2 and DR3. To achieve a physical prototype by the end of May 2026, our timeline accelerates the ME 450 deadlines, moving DR2 and DR3 up by two weeks and one week respectively. By DR3, we aim to finalize high-level software design and embedded hardware decisions following an initial round of iterative testing, ensuring proper mechanical integration and facilitating smoother system assembly toward the end of the project.

Table 1. Preliminary timeline of major project tasks. Tasks are organized into ME 450 Assignments, Control Tasks, and Vision tasks.

1	ME 450 Assignments	Start	End
1.1.1	DR1 Presentation	1/19/26	1/22/26
1.1.2	DR1 Report	1/23/26	1/29/26
1.2.1	DR2 Presentation	1/30/26	2/19/26
1.2.2	DR2 Report	2/20/26	2/26/26
1.3.1	DR3 Presentation	2/27/26	3/10/26
1.3.2	DR3 Report	3/11/26	3/17/26
1.4	DR4 Presentation	3/18/26	4/9/26
1.5	Design Expo	4/10/26	4/16/26
1.6	Final Report	4/17/26	4/26/26
2	**Control Tasks**		
2.1	Define Controls Scope	1/19/26	1/26/26
2.2	Order Preliminary Hardware Components	1/25/26	1/29/26
2.3	Hardware Rapid Prototyping	1/30/26	2/13/26
2.4.1	Design: High Level Software Architecture	2/14/26	2/24/26
2.4.2	Design: Program Navigation Module	2/24/26	3/8/26
2.4.3	Design: Program Control Actuation Module	2/26/26	3/8/26
2.4.4	Design: Program User Interface Module	3/9/26	3/16/26
2.4.5	Design: Integrate Full Controls Software	3/15/26	3/19/26
2.5	Control Internal Software Validation	3/20/26	3/29/26
2.6	System Integration and Debugging	3/29/26	4/10/26
2.7	Validation and Field Testing	4/8/26	4/15/26
3	**Vision Tasks**		
3.1	Define Vision Scope	1/19/26	1/26/26
3.2	Model Research and Selection	1/25/26	2/6/26
3.3	Turkey Vision Model Training	2/7/26	2/16/26
3.4	Select Camera and Supplementals	2/16/26	2/20/26
3.5	Design Final Vision Algorithm	2/18/26	2/27/26
3.6	Data Collection and Annotation	3/9/26	3/20/26
3.7	Vision Software Driver	3/21/26	3/27/26
3.8	Model Refinements and Debugging	3/28/26	4/10/26
3.9	Validation and Field Testing	4/8/26	4/14/26

6 DESIGN PROCESS

For the development of the computer vision and navigation systems, our team is following the Systematic Design Process integrated with the Socially Engaged Design (SED) framework [10] [11]. During the initial weeks of the semester, this process has focused heavily on the Problem Definition and Project Clarification phases. We have prioritized identifying the needs of all North Campus stakeholders, including students, faculty, and facilities to ensure our deterrent tactics will be effective and supported by the community. Through the SED framework, we have considered the broad impacts of deploying an autonomous robot in a public space, and have focused on safety and ethical considerations of human animal hazing.

The Hybrid V-Model and Agile approach are the most useful for this project. The autonomous robot requires a complex intersection of hardware (vision sensors/navigation computing) and software (machine learning models). A single linear model would be insufficient. The V-Model provides a structured framework for systems integration across the Drivetrain and Canine Design teams by mandating early verification of software to hardware interfaces. Agile iterations are used for the computer vision pipeline in the YOLO model and dataset. Additionally, developing the wireless UI through prototypes to benchmark streaming bandwidth and usability is following the agile methodology. This allows for continuous model refinement well before final hardware integration.

Our process differs from the standard sequential model by utilizing Concurrent Engineering and Front Loaded Validation [12]. For example, we are front loading hardware selection by benchmarking inference speeds against image resolution, to ensure we meet detection requirements without exceeding compute limits. Unlike the standard single team approach, we cannot wait for the physical concept selection to begin software development. We must work alongside the other subteams and use evolving placeholder requirements and models to ensure the navigation is ready for the finalized drivetrain. This process places more emphasis on user interface design and subsystem validation early on, while the standard model typically leaves validation for the end of the project lifecycle. This change is necessary to account for the risk of incompatibility in designs between the three subteams.

6.1 Information Sources

Our main source of information for engineering decisions is primary source interviews with stakeholders, such as Dr. Duquette, facilities, and community members. We have also done extensive research on turkey behavior and habits. Our technical specifications were informed by industry standards. Lastly, we followed academic literature and course modules from the University of Michigan engineering resources.

6.2 Engineering Standards

To ensure safe deployment on campus, our team is adhering to several industry standards that directly inform our system requirements and specifications. We are following ANSI/RIA R15.08-1-2020 [13] for the safety of mobile robots in shared human spaces, specifically regarding obstacle detection and emergency stopping. It informs requirement 2.1.2, mandating a 1m buffer from pedestrians, and requirement 3.1, which specifies the 2 second response time for emergency stop. ISO 13482:2014 [14] establishes safety requirements of service robots to prevent human injury. These mandates will inform our max speed in requirement 2.1.3 and deceleration performance in requirement 2.1.4. Additionally, our computer vision performance specifications are guided by IEEE 2671-2022 [15], which establishes standards for online detection and tracking systems based on machine vision. This dictates our YOLO model specifications, specifically our requirement for a minimum 20 pixel object height and 15 meter detection range. This ensures high confidence tracking before transitioning to chase protocol.

7 DESIGN CONTEXT

We split the project Stakeholders into three main categories. Primary stakeholders are directly impacted by the problem and/or the development of a solution. Secondary are part of the problem context but not directly impacted by the problem or solution. Tertiary are outside of the immediate problem context but may have the ability to influence the success or failure of a potential solution. In addition to weighting stakeholders based on severity of impact and influence, we categorized stakeholders to their roles in reference to the social context course learning modules [16]. The color coded list defines each role, and Figure 9 illustrates all categories below.

Resource Providers: provide financial, human capital, expertise, connections, and resources

Supporters & Beneficiaries of the Status Quo: benefit if the status quo is maintained

Complementary Organizations and Allies: influence in problem & solution

Beneficiaries and Customers: benefit from the solution in the problem space

Opponents and Problem Makers: contribute to the problem or oppose efforts

Affected or Influential Bystanders: no direct impact now, could become affected

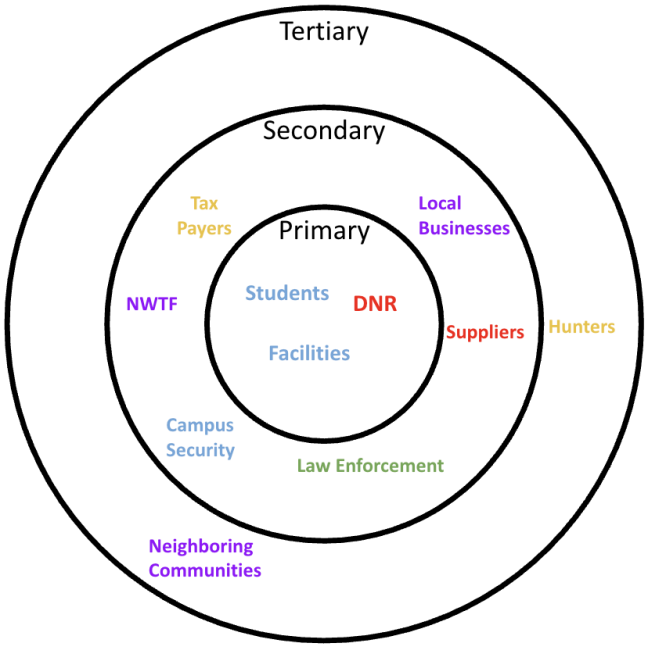


Figure 9. Stakeholder Map

The primary stakeholders include the students, DNR, and facilities. For the students, their primary interest is safety and separation from wildlife. Their agency is feedback in surveys that will influence design choices and location. Facilities have an interest in reducing the number of complaints and property damage from the birds. They have influence with how the solution is implemented as the end user of our prototype. The DNR's main goal is to reduce human wildlife conflict, and potentially use our design for a larger scope in other areas. Secondary stakeholders include groups such as security, law enforcement, along with suppliers and local businesses.

7.1 Intellectual property

This project is uniquely structured due to our partnership with the DNR. Unlike typical university sponsored research projects, we are not working as employees and have external sponsorship. According to the University of Michigan Standard Practice Guide (SPG 303.04), the University does not claim ownership of IP created by students in the furtherance of curricular classwork or capstone projects [17].

7.2 Sustainability

Our teams aim for long term sustainability. To ensure our solution is sustainable, we must consider the full cradle to grave lifecycle of the system. Manufacturing the robot requires material consumption, most notably lithium ion batteries and rare earth metals for the motor. Extracting materials like cobalt and lithium creates a big environmental and social footprint. During use, the robot will have zero emissions during operation, however its electricity will be

drawn from the local grid. To increase sustainability, we will consider using recycled materials and look at implementing solar assisted charging. The deterrent may also have negative social impacts, such as excessive barking noise and getting in the way of pedestrians and vehicles. To reduce the possibility of these consequences, we will work with the community by interviewing members and taking feedback on design choices and results. Lastly, there are economic hurdles to reach sustainability. The cost to produce our robot at scale would exceed the budget of $10,000, since there won't be a group of 13 unpaid engineering students to build them. Communities will not be able to afford the robot unless it is produced in high quantities to reach the economies of scale for material and manufacturing costs.

7.3 Ethical Considerations

Managing wildlife in a high traffic urban environment requires a design that balances human safety with animal welfare. We have grounded our ethical framework based on guidelines of our sponsor and national conservation leaders. According to the DNR, control measures must only be implemented when damage or a nuisance is both "necessary and warranted" [18]. Following this ensures the robot only executes hazing tactics specifically for safety and property concerns, rather than hazing indiscriminately. Additionally, we will uphold the NWTF value of Healthy Habitats and Health Harvests, which requires that our operations do not disrupt the overall health of the turkey population [19]. By utilizing motion and sound instead of physical contact or force, the design satisfies the ethical requirement to minimize stress during hazing. We also will continue to take feedback from the community and facilities to ensure the device is not negatively impacting members.

8 USER REQUIREMENTS AND SPECIFICATIONS

Project requirements are organized into four primary categories to guide prototype development: Environment Detection, Movement, User Functions, and Audio. Each category is structured hierarchically, with sub-requirements grouped by related functionality (for example, Requirements 1.1.1 and 1.1.2 both address vision accuracy). The full set of requirements and specifications is presented in Table 2. Justifications for each requirement and the corresponding validation procedures are discussed in the subsequent sections.

Table 2: Complete list of user requirements and specifications. Requirement priority of 1 is the highest and 3 is the lowest.

Requirement Number	Requirement Name	Priority	Specification
1. Environment Detection			
1.1.1	Accuracy: Turkey Detection	1	While a turkey occupies > 80% of its body area within the camera field of view, and resides from 0-15 m away, the system shall correctly classify > 90% of those frames as containing a turkey. The model shall also falsely identify non-turkey objects as turkeys in < 3% of cases.
1.1.2	Accuracy: Bounding Box	1	The associated intersection over union score (IoU) > 0.7. IoU is defined as the area of intersection divided by the area of union between ground truth bounding box and predicted bounding box.
1.2.1	Speed: Zone Scan Time	2	The system must be capable of providing coverage of the full zone, with entire zone coverage observed at least once every 10 minutes.
1.2.2	Speed: Maximum Latency	2	While classifying an image, the system shall not exceed a maximum classification latency of 1 second per image.
1.2.2	Speed: Computing Resources	3	While classifying an image, the system shall not consume more than 50% of available CPU resources or more than 70% of available memory.
2. Movement			
2.1.1	Safety: Zone Confinement	1	The system shall maintain a minimum separation distance of 2 meters from all non-roadway boundaries and a separation distance of 4 meters from the southern roadway boundary.
2.1.2	Safety: Obstacle Avoidance	1	The system's maximum traversal speed shall not exceed 0.5 m/s when operating within 4 meters of any detected human. The system shall maintain a minimum separation distance of 1 meter from all identified humans and obstacles at all times.
2.1.3	Safety: Maximum Travel Speed	2	The maximum traversal speed of the system shall be less than 3 m/s while pursuing a turkey and less than 2 m/s when no turkey is present

Requirement Number	Requirement Name	Priority	Specification
2.1.4	Safety: Deceleration Time	2	While operating at maximum traversal speed, the system shall be capable of decelerating to below 0.05 m/s within 2 seconds.
2.2.1	Turkey Deterrence: Rate of Displacement	1	While the system is pursuing a turkey, the turkey's average displacement rate out of the zone shall be at least 0.5 m/s once within < 5 m from the turkey.
2.2.2	Turkey Deterrence: Bounded Pursuit	2	While the system is pursuing a turkey, the turkey shall not travel more than 4 m toward the center of the zone, and the total pursuit distance shall not exceed 40% of the maximum between the zone's width and height once within < 5 m from the turkey.
2.2.3	Turkey Deterrence: Zone Coverage	2	The system shall be able to traverse the entire zone with a positional resolution of 0.5 m and an angular resolution of 10°.
3. User Functions			
3.1	Manual Input Latency	1	The user shall be able to provide manual motor inputs with a maximum latency of 1 second. The user shall also have access to an emergency stop that reduces the device's speed to zero in less than 2 seconds
3.2	Device Tracking	2	The user shall have access to the device's geolocation with an error less than 1 m. The user shall also have access to system data that is updated at least once per minute
3.3	Video Display	3	The user shall have access to a live video feed from the device with a minimum resolution of 240p and a minimum frame rate of 5 fps.

8.1 Environment Detection Requirements

A major responsibility of the system and this subteam is the development of a vision system capable of reliably identifying turkeys while also detecting and avoiding distractions such as pedestrians, vehicles, and other obstacles. Because the device operates in real time, the design must balance detection accuracy with available computational resources. Accordingly, the vision requirements are divided into two groups: accuracy requirements (1.1.X) and classification requirements (1.2.X), which are outlined below.

8.1.1 Accuracy

Having a reliable vision system is imperative to the safety and effectiveness of our system. To accomplish this, our team needed to develop measurable targets that we could use to evaluate the accuracy, speed and safety of our computer vision model. Most crucially, we have determined using a previously implemented model for this project, SpeciesNet, that an accuracy of 90% for true positive identifications on turkeys was both realistic and would allow our robot to be effective. An example of true positive detections using the previously used model can be seen in Figure 10 below. Conversely, while it is less important for us to correctly identify other objects such as humans, we determined that it was essential for us to remain under a 3% threshold of false positive turkey identification. Incorrectly targeting other animals like dogs, or small children, could pose a safety concern or at the very least distress the general public. One change to our specifications that was made from our DR1 is decreasing our detection radius from 20 m down to 15 m. This revision was due to the change in our system approach, from a stationary sentry to the roaming robot. Given that our robot can now utilize movement between waypoints to cover a larger zone, the longer detection distance is no longer necessary, and reducing it to 15 m also demands a lower performance camera.

Another important measure of accuracy is how well our model is able to create a bounding box around the targets. This is a necessary consideration as poor bounding boxes result in poor object tracking, and the eventual integration of the robot's control response will be less accurate as well. To quantify the quality of our bounding box, our team researched standards that are used currently in industry, and determined that the Intersection Over Union (IoU) would be the most useful and easily applicable model assessment metric [20]. In essence, by taking the quotient of the area of overlap over the area of union between the bounding box and the ground truth box, we can get a sense of how large our localization error is when detecting objects. While different use cases demand different levels of accuracy, we believe that a score of above 0.7 is achievable and would allow our robot to function effectively.

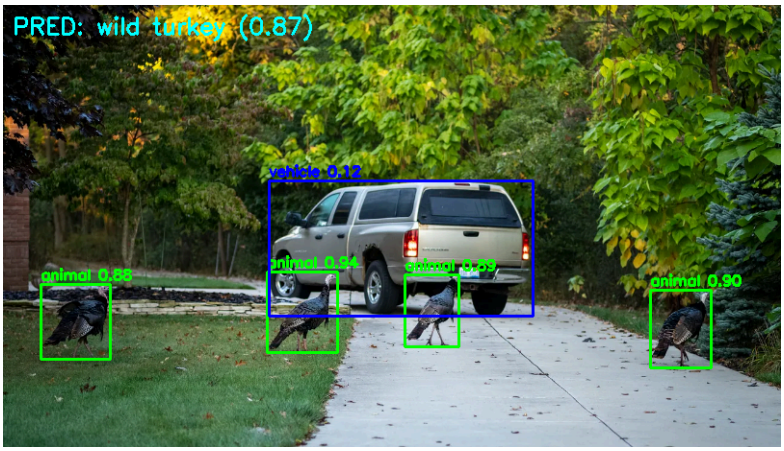


Figure 10. Object detection and prediction scores of turkey image sample, using the previously implemented SpeciesNet model.

8.1.2 Speed

To ensure the effectiveness of our robot, it is essential that our vision system is fast enough in identifying, and responding to turkeys in the environment. The first metric to assess is how fast we can scan the selected zone, so that our response is timely and actually prevents disturbances. Given our switch to the roaming robot, we have overhauled the previous scan time specification, and instead we are requiring our solution to be capable of scanning the defined zone within 10 minutes. Due to our adjusted 15 m detection range, we believe that 10 minutes is a conservative estimate of the maximum time it would take to patrol the zone, while still remaining effective as a deterrent.

Another important assessment of our model is how fast it can process each image, and make predictions within a fixed time. After running initial tests on the SpeciesNet model, we decided that a time of less than 1 second per image is realistic, and would allow us to still make accurate predictions while scanning the environment. We plan to also train a CV model using YOLOv8, and will evaluate which model gives us the best results in the shortest amount of time.

Lastly, we also need to consider the computing resources of the onboard systems. While we have yet to determine exactly what hardware will be used, utilizing less than 50% of the CPU resources and less than 70% of the available memory will give us the appropriate resources to make predictions, without straining our equipment and slowing down the computation process.

8.2 Movement Requirements

A primary component of the final system is autonomous mobility. Given the presence of trees and other hazards in our zone, we need to balance effective turkey deterrence with safe pathing.

Accordingly, the movement category of requirements is divided into two categories: Safety (2.1.X) and Turkey Deterrence Requirements (2.2.X), which are detailed below.

8.2.1 Safety

Safe operation within the designated zone is a primary requirement. Our defined zone includes a few stationary collision hazards for our system, including trees, buildings, poles, and an abstract statue. We also need to prevent collisions with dynamic entities inside our zone, including pedestrians, bikers, and non-turkey wildlife. The system must also remain within the specified zone boundaries at all times, for an escape could lead to unpredictable and dangerous outcomes.

Requirements 2.1.1 and 2.1.2 ensure, at the algorithmic level, that the system does not plan or execute unsafe behavior. Requirement 2.1.1 establishes a minimum separation distance of 4 meters from the southern boundary at Hayward Road and 2 meters from all other boundaries. The increased buffer along Hayward Road reflects the elevated risk associated with entering a roadway and potential vehicle collisions. The 2-meter buffer for the remaining boundaries was selected based on the maximum speed and deceleration limits defined in Team 18's drivetrain requirements, ensuring adequate stopping distance within the operational zone.
To account for unexpected disturbances and system error, Requirements 2.1.3 and 2.1.4 impose limits on maximum vehicle speed and allowable stopping time. Requirement 2.1.3 references the 3 m/s maximum speed specified in Team 18's drivetrain design, while a reduced operational limit of 2 m/s is established to provide sufficient reaction time for nearby individuals if the system deviates from its intended path. The deceleration requirement defines a maximum stopping time to ensure that both remote and autonomous emergency stop mechanisms can prevent or mitigate a potential collision.

8.2.2 Turkey Deterrence

In addition to safe operation, overall system effectiveness depends on successfully deterring turkeys from the designated zone. Requirement 2.2.1 specifies a minimum displacement rate to ensure active deterrence. A threshold of 0.5 m/s was selected to elicit a measurable turkey response while still allowing the system to exhibit controlled behaviors, such as gradual approach and "pointing," consistent with Team 19's operational objectives. A maximum displacement of less than 5 meters was established to prevent excessive pursuit in cases where a turkey is detected at a significant distance. This constraint ensures the system initiates and executes a deterrence sequence without compromising boundary compliance or safety requirements.

Battery endurance during operation is a primary constraint. To reduce energy consumption and limit maintenance intervals, Requirement 2.2.2 bounds the pursuit duration once the system is within 5 meters of a detected turkey. This limit prevents extended chases that provide minimal additional deterrence while increasing path inefficiencies and reducing overall zone coverage

effectiveness. An additional constraint limits inward displacement to less than 4 meters toward the center of the zone during pursuit. This requirement constrains path geometry and reduces the likelihood of turkeys remaining within the zone while being chased, thereby improving deterrence efficiency and conserving battery resources.

To address the presence of fixed obstacles and narrow chokepoints within the operational zone, Requirement 2.2.3 specifies minimum maneuverability capabilities for confined-space traversal. A linear resolution of 0.5 meters and an angular resolution of 10 degrees were selected as the minimum thresholds necessary to navigate constrained areas without excessive backtracking or path inefficiency.

8.3 User Functions Requirements

Integration with the systems developed by Team 18 and Team 19 presents elevated risk. To mitigate safety concerns, the system shall include a remote emergency stop capability. Requirement 3.2 specifies that the user interface must enable a remote emergency stop with a maximum response time of 2 seconds. This duration is aligned with Requirement 2.1.4 and may necessitate aggressive deceleration to meet stopping constraints.

For validation and debugging purposes, Requirements 3.3 and 3.4 mandate real-time geolocation reporting and an active video feed. These capabilities are necessary to monitor system state, diagnose integration issues, and verify operational behavior during testing. The specified thresholds represent the minimum performance levels required to provide adequate situational awareness.

9 CONCEPT GENERATION AND SELECTION

After outlining the design requirements and specifications, our next step was to begin generating concepts for our solution, and narrow them down to a final design. To accomplish this, our team was in constant communication with the two other sub teams, in order to understand what approaches were feasible within the scope of the project, not just for our own tasks, but the team as a whole. We would like to note that, as our team is focused mostly on computer vision and low level controls, this stage of the design process looked quite different from conventional concept generation processes, and focused more on the system concept and software/hardware selection.

9.1 System Navigation Options

Team 17 is responsible for finding the most optimal navigation protocol to meet our sponsors objective of humane deterrence. To aid in concept generation, we created a morphological chart with design functions of locomotion, detection sources, sensing coverage, deterrence, navigation, and computation, seen in Table 3 below.

Table 3. Morphological Chart of System Strategy

Design Function	Option 1	Option 2	Option 3
Locomotion	Mobile Zone	Autonomous	Stationary
Detection Source	Onboard	Onboard & Grid	Distributed Grid
Sensing Coverage	Localized	Full Zone	Fixed
Deterrence Logic	Search & Chase	Reactive Interception	Proximity
Navigation Mode	Path Planning	Vectoring	Static Placement
Computation	Centralized	Distributed	Decentralized
Primary Limitation	Blind Spots	Complexity	Habituation Risk

After creating over 10 options with different combinations of design functions, three concepts were established for navigation strategy and scare tactics: the Sentry Dog, the Responder Dog, and the Stationary Swarm. Each concept represents unique trade offs between mechanical complexity, computational load, and spatial coverage. Figure 11 illustrates each idea.

a. b. c.

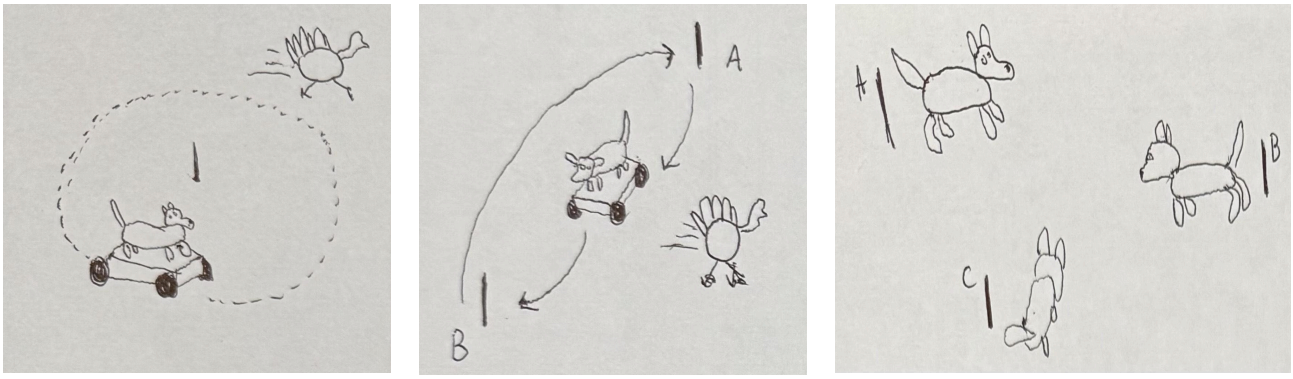


Figure 11a, b, c. Three concepts for system navigation and scare strategies for Sentry, Responder and Stationary Swarm, respectively.

The Sentry Dog is a singular mobile platform with a wander and scan methodology. The device continuously traverses a designated zone while the vision system checks for targets. This architecture combines vision processing and navigation into one hardware package, eliminating external infrastructure and communication. One drawback is temporary blind spots, since the robot can’t view multiple directions with a single camera.

The responder dog separates detection from deterrence. It uses a grid of stationary waypoints with cameras monitoring different areas on campus. When a waypoint detects turkeys, the dog will patrol to that area. The responder dog provides a larger zone coverage and eliminates patrol gaps. Disadvantages are increased cost and complexity for separate waypoints and navigation protocol. The robot would need to identify and avoid obstacles, pedestrians, and traffic.

The swarm consists of multiple stationary units. It is the least complex system with no drivetrain or navigation. Each dog would be equipped with cameras and have a movement mechanism and audio to mimic a canine. The swarm requires more material and labor to set up multiple dogs, which increases cost. Additionally, turkeys are known to habituate to non threatening decoys. Because of the predictable motion, turkeys would likely learn to ignore the scare tactics.

To compare the three options quantitatively, the group created a Pugh chart with weighted criteria in order of deterrence, spatial coverage, cost, complexity, and zone flexibility, seen in Table 4 below.

Table 4. Pugh chart for system strategy concepts.

Criteria	Weight	Sentry Dog	Responder Dog	Stationary Swarm
Deterrence	4	0	0	-1
Spacial Coverage	3	0	1	1
Cost	2	0	-1	-1
Complexity	2	0	-1	1
Zone Flexibility	1	0	0	-1
Total Score	13	0	-1	-2

The Sentry Dog emerged as the optimal concept due to its balance between deterrent capability and mechanical simplicity. While the Responder dog offers superior spatial coverage, it falls short in complexity with more hardware and navigation protocol. The Stationary Swarm was eliminated due to its poor deterrence and high logistical costs.

9.2 Camera Mounting

In addition to our system concept generation, our sub team was also responsible for generating concepts for our camera mount, as it would have a direct impact on our approach to scanning, and how suitable the images were for detection purposes.

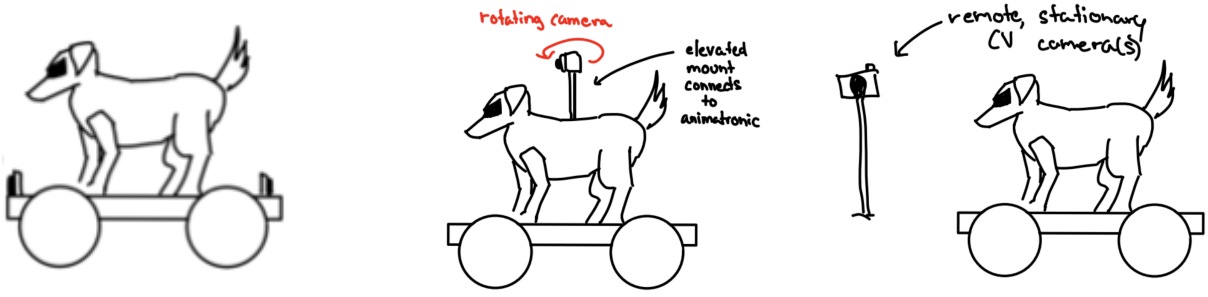
To explore possible camera mounting options for our computer vision system, our team utilized a morphological chart, breaking down the mount into primary design functions, which can be seen in Table 5 below. The key functions that we chose to consider included camera position, number of cameras, mounting mechanism, panning/rotation capabilities, and height/elevation of the mount. These functions were selected as they directly impact the image processing procedure, and can affect scan time and our ability to detect turkeys within the zone.

For each design function we generated at least 4 possible options, and began combining them to allow for rapid concept generation.

Table 5. Morphological chart for camera mount design functions.

Design Function	Option 1	Option 2	Option 3	Option 4
Camera Position	Dog head	Top of dog	Extendable mast	Fixed remote camera
Number of Cameras	Single camera	Front and rear	3-4 cameras	1 per waypoint
Mounting Mechanism	Fixed bracket	Magnetic quick release	Gimbal for stabilization	Pan-tilt servo
Pan/Rotation	None (static)	Stepwise 360° motor rotation	Cont. 360° motor rotation	180° wiper coverage
Height/Elevation	On cart	Below cart	Robot eye-level	Elevated mount (mast)

Once we had established a wide array of possible concepts, we began narrowing them down to 3 primary designs, based on mount complexity, image processing complexity, and zone coverage capacity. These options can be seen in Figure 12 below.



a. **b.** **c.**

Figure 12a, b, c. Concept 1, 2 and 3 for computer vision camera mounting, respectively.

Concept 1, seen in Figure 12a above, consists of a camera, or multiple cameras which are mounted directly to the baseplate of the robot. This approach simplifies the camera mounting process, while having the highest stability. The drawback would be that a single camera, without panning capabilities, is insufficient in detecting a 360 degree FOV, and therefore would need to be compensated for either using multiple cameras, or the movement of the robot.

Concept 2, seen in Figure 12b above, consists of a mast that elevates the camera above the animatronic, with a servo motor for full 360 degree rotation. This design allows for full unobstructed coverage of the surroundings with a single camera, simplifying the image feeding process. However, it will require integration with team 19's animatronic design, as well as a separate and potentially unstable mast for mounting.

Concept 3, seen in Figure 12c above, remotely mounts multiple CV cameras at key locations within the zone. When a turkey is detected, the canine robot patrols to a predetermined waypoint in order to deter them away. This option results in highly stable cameras as they are stationary, but creates complications in translating detected objects into known coordinates in the zone.

After evaluating each design and comparing their advantages and drawbacks, we determined that concept 1 would be our selected approach, with a single front camera. Multiple cameras was simply not a viable approach, as it required feeding additional images into our model, and slowing down the processing speed. For example, 2 images taken from 2 different cameras at the same time would need to be processed separately, resulting in double the processing time. We will instead use the movement of our drivetrain to strategically cover the entire zone, and ensure that turkeys are unable to idle within the zone for extended periods of time.

10 HARDWARE & SOFTWARE SELECTION

Given that our team is primarily responsible for low level controls and the computer vision system, a large part of our design process is dedicated to researching and selecting the most appropriate hardware components for our specific solution, as well as selecting the most suitable computer vision model.

10.1 Computer Vision Model Selection

A computer vision model is an essential component of our design because we require our camera not only to detect the turkeys but also locate them. We listed several commonly used algorithms for computer vision in the Pugh chart in Table 6. The main criteria for model selection is precision, running speed in microprocessor, computation cost for training, detection robustness, and algorithm architecture complexity.

After reviewing several computer vision models, we have chosen Complete Pipeline (SpeciesNet) and One-Stage CNN (YOLOv8) as our primary options. This part of the report details the technical analysis behind these selections and how they meet the requirements of our system.

Table 6. Pugh Chart for computer vision model selection. The One-Stage CNN (Yolo Family) and Complete Pipeline (SpeciesNet) are considered as suitable models.

Criteria	Weight	Classical CV	One-Stage CNN	Mask R-CNN	Complete Pipeline	Transformer -based
Precision	5	0	1	1	2	2
Speed	3	0	0	-1	-1	-2
Computing Cost	2	0	-1	-1	-2	-2
Robustness	1	0	1	1	2	2
Complexity	2	0	-1	-2	-2	-2
Total Score	13	0	**2**	-3	**1**	-2

SpeciesNet is a complete computer vision pipeline developed by Google Research in collaboration with wildlife biologists, mainly used in detecting and classifying wildlife captured by camera trap images. With its advanced three-stage pipeline (detector + classifier + post-processor) and millions of images from camera traps worldwide as training data, SpeciesNet is capable of detecting and classifying animals in different environments with high precision. In addition, the built-in detector can also detect daily objects such as vehicles and pedestrians. However, the primary drawback of SpeciesNet in the context of our system is its lack of architectural flexibility. Because it is a proprietary, pre-trained framework designed to identify hundreds of different species globally, it cannot be customized for a single-target application. This results in significant computational redundancy, as the model spends many resources through classification layers that are irrelevant to our use case. When running the model in a microcontroller, this will lead to a large latency in detection (<1 fps approximately).

On the other hand, YOLO26 (You Only Look Once) was selected for its balance of speed and modularity, making it ideal for deployment on a microcontroller. Unlike the broad taxonomic approach of SpeciesNet, YOLOv26 features a modular architecture that allows for Deep Task-Specific Pruning. This enables us to strip away the weights and layers associated with thousands of irrelevant categories, focusing the model's entire discriminative power solely on the

detection of wild turkeys in urban environments. By utilizing Quantization-Aware Training (QAT) and a significantly reduced parameter count, YOLOv26 can achieve near-instantaneous inference speeds on low-power microcontrollers. This efficiency ensures that the robot can maintain a high frame rate, allowing for the real-time spatial tracking and immediate deterrent response necessary to handle fast-moving targets in a dynamic neighborhood setting.

The ultimate selection between these two architectures will be determined by a benchmark of detection precision against real-time performance. Our primary objective is to deploy a custom-trained YOLOv26 model. If our localized training on urban turkey datasets can achieve a precision threshold comparable to the baseline established by the more robust SpeciesNet in our specialized scenarios, YOLOv26 will be the definitive choice for the final deployment. However, should the specialized pruning and quantization of YOLOv26 lead to unacceptable accuracy degradation or missed detections in complex urban backgrounds, SpeciesNet will serve as our high-reliability backup.

10.2 Camera Selection

Even with a highly trained computer vision model, our ability to detect turkeys is also heavily dependent on the capabilities of our camera, and how compatible they are with the model as well as the on-board microcontroller. Our team explored a wide range of camera types and brands, using the following set of criteria: detection capacity at 15 meters, lens flexibility, microcontroller compatibility, image resolution, processing efficiency and cost. Based on this criteria, the top candidates were compiled and summarized in the Pugh chart in Table 7 below. The Raspberry Pi Camera v3 was used as our baseline, as it is a widely used option for simple computer vision projects, and it is also extremely cheap, so we can use it for testing if needed.

Table 7. Pugh Chart for computer vision camera selection. The chosen model is the Raspberry Pi HQ, with interchangeable lens support.

Criteria	Weight	Pi Camera v3 (baseline)	Raspberry Pi HQ (with lens)	Arducam IMX477 HQ (with lens)	Luxonis OAK-D Lite (USB cam)
Detection @ 15 m	4	0	1	1	1
Lens flexibility	3	0	1	1	0
RPi compatibility	2	0	1	0	-1
Image resolution	2	0	1	1	1
Processing efficiency	1	0	0	0	1

Cost	1	0	-1	-1	-1
Total Score	13	0	**10**	8	4

For our final selection, the two camera options that we thought were most appropriate for our purposes were the Raspberry Pi HQ and Arducam IMX477 HQ, with interchangeable lenses. Both gave us the necessary image resolution and distance capabilities for our project, and were quite similar in terms of specifications. Ultimately, we decided to go with the Raspberry Pi HQ, due to superior compatibility with our Raspberry Pi 5 microcontroller. Despite this evaluation, further field testing will be the only way to determine if the camera is sufficiently effective, and what lens we should pair the camera with.

10.3 Navigation Selection

Deciding on our approach to the navigation system is key to the functionality of our canine robot, as it directly impacts our ability to traverse the zone, avoid obstacles, and approach turkeys. In order to evaluate our options, we came up with the following set up criteria, determined by metrics that were either essential to our design functions, or significantly impacted accessibility: positional accuracy, outdoor robustness, integration, mechanical setup, and financial cost. Based on this criteria, the top candidates were compiled and summarized in the Pugh chart in Table 8 below. AprilTags were selected as our baseline as they provided the necessary positional accuracy for our project, while also satisfying the rest of our design criteria to an acceptable standard.

Table 8. Pugh Chart for navigation system selection. The chosen option is the UWB anchors and tags.

Criteria	**Weight**	**AprilTags (baseline)**	**Geofencing**	**Accelerometers**	**UWB Anchors**
Positional Accuracy	5	0	-1	-1	1
Outdoor Robustness	3	0	1	-1	1
Integration	2	0	1	0	0
Setup	2	0	1	-1	0
Cost	1	0	1	1	-1
Total Score	13	0	3	-9	**7**

Based on our evaluation, the UWB anchors were by far the best solution for our project. Although AprilTags are a well-rounded solution, we believe that ultimately the UWB anchors as well as the geofencing solution provide superior performance in the outdoor environments we

are working in. Geofencing had the advantage of easy integration, low cost, and a plethora of existing solutions to model off of, but lacked the necessary positional accuracy from preliminary testing. On the other hand, UWB anchors delivered on every design criteria, at the expense of a higher financial cost. Despite this, we believe that it is a necessary investment for the success of our project, and as such, this was our selected navigation solution.

11 PRELIMINARY DESIGN CONCEPT

The selected design centers on the Sentry Dog, a mobile platform that highlights mechanical simplicity and autonomous scanning in a predetermined zone. The hardware suite features a Raspberry Pi HQ camera mounted directly to the baseplate of the robot, utilizing the drivetrains rotation to achieve spatial coverage. For real time detection, we selected a pruned YOLOv26, with SpeciesNet serving as a backup. The vision model will run on a Raspberry Pi 5. Navigation is managed through UWB anchors, which provide optimal positional accuracy and outdoor durability to ensure the system tracks and deters targets. The hardware choices are visually summarized below in Figure 13.

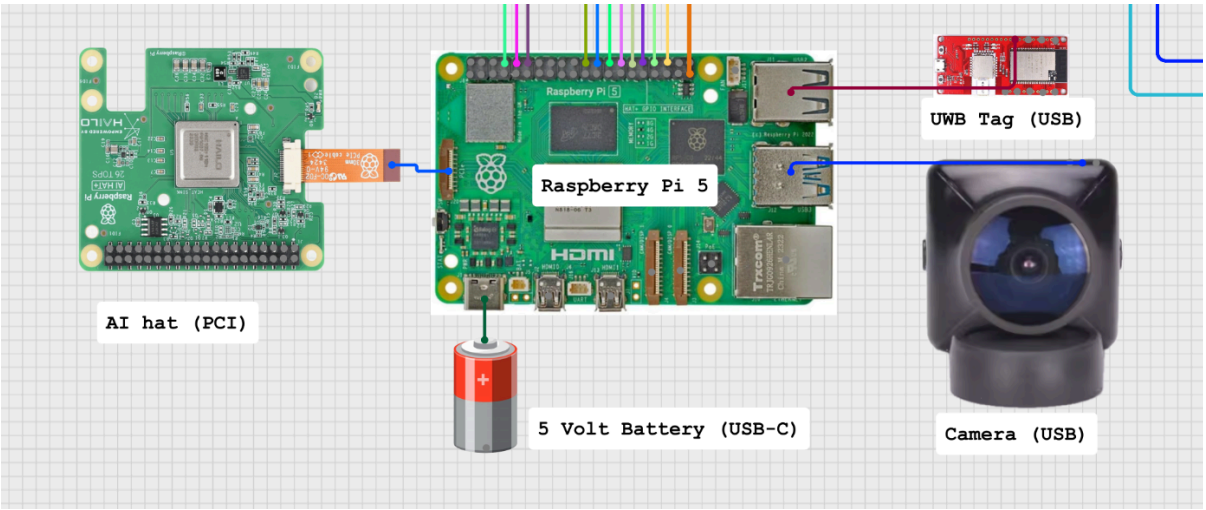


Figure 13. Wiring Diagram of Navigation & Vision system connected to Raspberry Pi

The team has developed a software architecture structured as a finite state machine with three operating modes: Scanning Protocol, Alert Protocol, and Chase Protocol. In the Scanning Protocol, the robot navigates to predefined waypoints selected to maximize coverage of the operational zone and performs controlled angular sweeps to detect turkeys and humans within its field of view. Upon detecting a turkey, the system transitions to the Chase Protocol, during which the robot executes deterrence behaviors to drive the turkey out of the zone. After the turkey exits the zone boundaries, the system enters the Alert Protocol, maintaining heightened monitoring to prevent re-entry while allowing continued retreat; if no turkey is detected for a specified timeout period, the system returns to the Scanning Protocol. A diagram of the states with transitions is shown below in Figure 14.

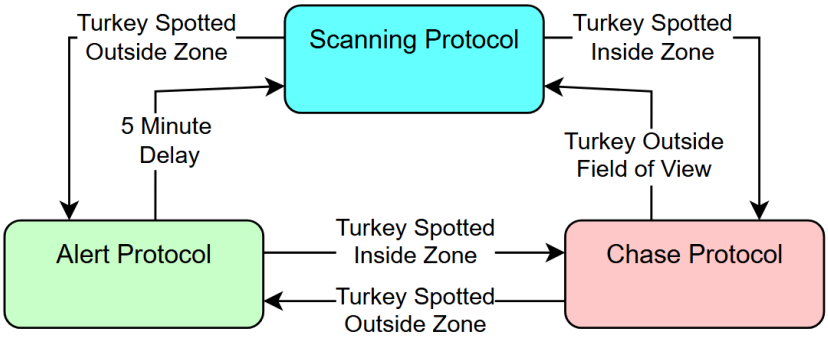


Figure 14. High Level State Concept. The Scanning, Chase, and Alert protocol are shown in blue, red, and green respectively and are color coordinated for similar subsequence figures.

11.1 Scanning Protocol

Within the Scanning Protocol, the system performs two primary tasks: navigating to the next waypoint along a preplanned route and conducting a full-area scan once the waypoint is reached. Because the camera cannot capture a complete 360° field of view from a stationary position, the robot rotates its drivetrain in a stop-and-go pattern to cover the surrounding area. A state diagram of this scan subprotocol is shown below in Figure 15. The number of rotations at each waypoint and the delay between scans can be adjusted based on power consumption requirements and the available energy supply.

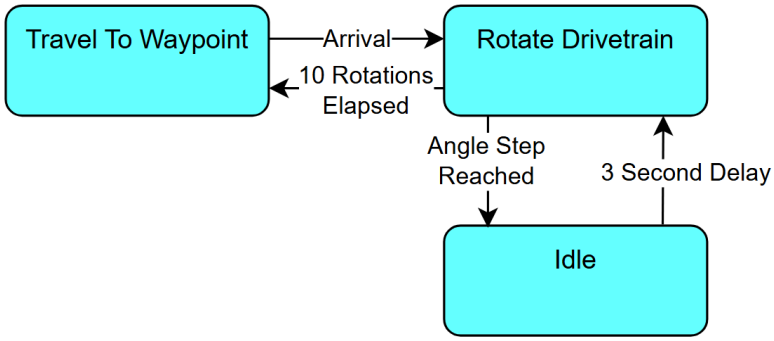


Figure 15. The Scanning Subprotocol outlined in blue. The three primary states are Travel To Waypoint, Rotate Drivetrain, and Idle.

To implement the Travel-to-Waypoint state, we first identify optimal scanning locations within the zone to maximize coverage while minimizing the number of waypoints and required actuation at each. These locations will be refined through additional testing to validate their effectiveness, but a preliminary set of zone waypoints for scanning is shown in Figure 16.



Figure 16. Preliminary zone definition with waypoint identified with the purple letters. The sequence will traverse alphabetically, looping at the end from G to A

11.2 Chase Protocol

Once the vision system detects a turkey within the zone, the system transitions to the Chase Protocol. This mode employs path generation and trajectory tracking algorithms to avoid static and dynamic obstacles while ensuring effective turkey deterrence. The software architecture supports dynamic trajectory updates, but further specifications from Team 19 are required to determine the timing and selection of canine actuators. A state diagram of the Chase subprotocol is shown below in Figure 17.

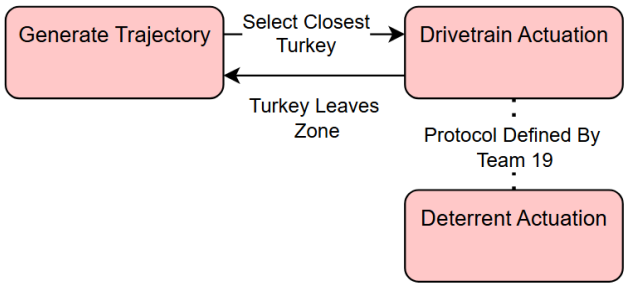


Figure 17. A diagram showing the chase subprotocol. Significant canine-related design elements are still needed from Team 19 which is outlined by the dotted line.

11.3 Alert Protocol

Once a chased turkey exits the zone, the system remains in the Alert Protocol to continue deterrence through coordinated audio cues and micro-movements. This ensures that turkeys maintain their retreat and do not re-enter the zone if the robot were to turn away immediately at the boundary. The protocol integrates audio deterrents with small drivetrain adjustments to keep the camera centered on the turkey and maintain line of sight. A schematic of the Alert subprotocol is shown in Figure 18.

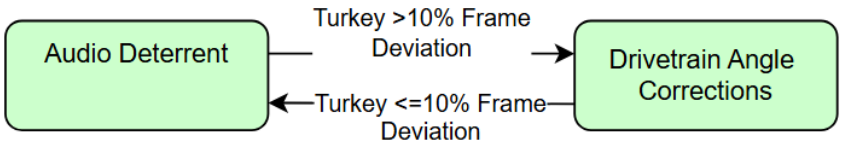
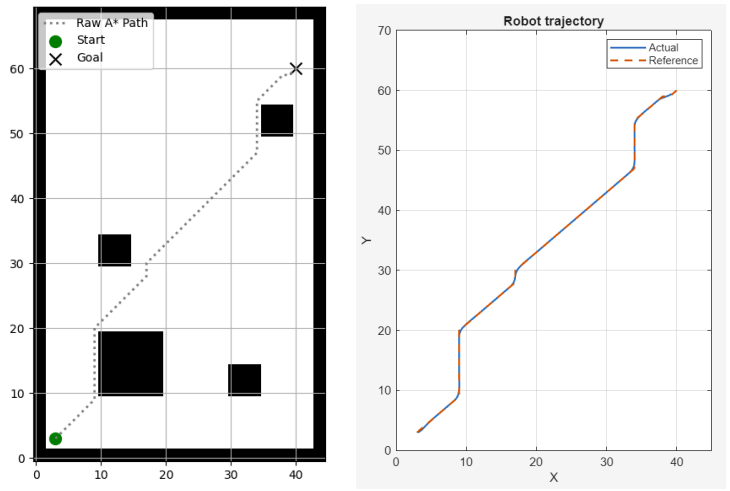


Figure 18. A diagram showing the alert subprotocol. The 10% Frame Deviation is a placeholder once we identify an appropriate deviation to induce drivetrain angle corrections.

11.4 Path Generation and Trajectory Tracking Algorithm

Traversing the zone while avoiding obstacles is a critical aspect of our design and addresses requirements 2.1.1, 2.1.2, and 2.1.3. In our preliminary approach, we represent the zone as a grid using the Python library NumPy and implement an A* algorithm to compute locally optimal paths from point A to point B while avoiding obstacles. A sample output of this pathfinding process is shown in Figure 19A.

The output from the A* algorithm provides a path, but has no mechanism for driving the control actuator to get there. Hence, we need another middle level algorithm to convert navigation paths to control inputs. We will use a Model Predictive Control algorithm to track the reference path and communicate with our low level control driver. A plot of this is shown below in Figure 19B.



(A) **(B)**

Figure 19. (A) shows an example A* algorithm with black boxes representing obstacles, green representing starting location and the x representing the end destination. (B) shows this same pathway being simulated in matlab using the MPC algorithm. The reference and actual plots are shown in red and blue respectively.

While the details of our implementation of these algorithms are still being developed, they provide a proof of concept for implementing obstacle avoidance and maximum speed enforcement in our final deliverable.

11.5 User Interface

To fulfill requirements 3.1, 3.2, and 3.3, we have constructed a preliminary wireless user interface with some preliminary data for proof of concept. This user interface is shown below in Figure 20.

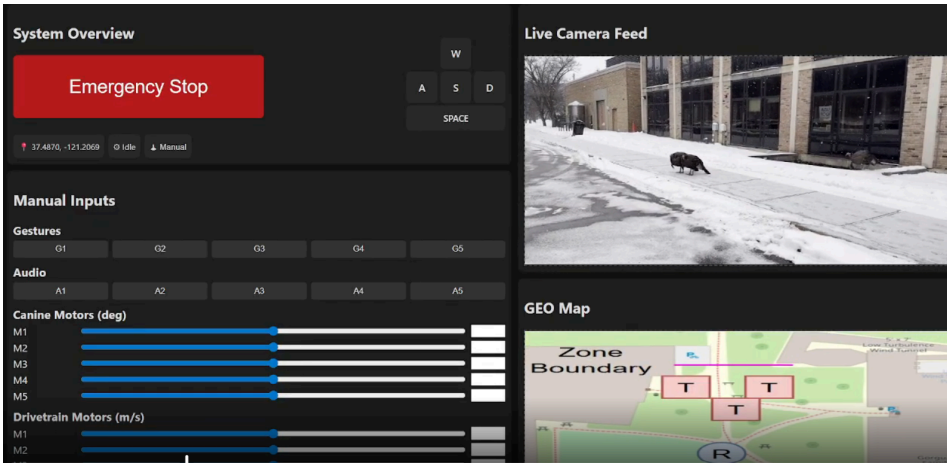


Figure 20. Preliminary design concept of user interface. Emergency stop is shown in red on the top left, streaming of a video is shown on the top right, and a stop-motion placeholder depicting the GPS positioning is shown in the bottom right corner.

Although the full communication protocol is still under development, the current prototype supports the most bandwidth-intensive operation, video streaming. The streamed video exceeds the resolution and frame rate required by the system, providing a performance benchmark. Additional field testing will be conducted under lower-quality service conditions to verify the interface maintains functionality across varying operational scenarios.

12 FINAL DESIGN DESCRIPTION

Following the preliminary design review, the software architecture has been refined and streamlined to improve implementation clarity and debuggability. Notable developments include enhancements to the computer vision algorithm, the introduction of a turkey distance estimation algorithm, and progress on mechanical design including CAD models for the camera and UWB sensor housings. A preliminary testing prototype has also been constructed to facilitate software validation prior to integration with Team 18 and Team 19. This section outlines the current software architecture, individual module designs, mechanical developments, and wiring plans.

12.1 Software Architecture

The software architecture employs a publish-subscribe model, in which discrete modules operate in isolation and communicate through defined data channels. Receiver functions subscribe to

these channels and execute their self-contained logic upon receiving published data. The architecture consists of seven distinct modules, with brief descriptions provided in Table 9 and a schematic of their interconnections presented in Figure 21 below.

Table 9. Overview of system modules with their associated protocols and descriptions. Entries are color-coded by module, with striped rows indicating corollary protocols.

Module	Protocol Name	Color	Description
Vision	VisionObject	Red	Turkey and human objects in frame with X,Y and relative theta estimates.
Vision	VideoFeed	Red-Striped	Live camera feed with bounding box and object classification annotations.
Canine	Canine	Orange	Servo pose and audio commands for robotic canine.
Canine	ServoFeedback	Orange-Striped	Servo angle feedback from canine for use in user interface.
Drivetrain	DriveCommand	Green	Motor speed commands to be processed in the drivetrain.
Drivetrain	SlipFeedback	Green-Striped	Feedback protocol to indicate estimated current slip for use in user interface.
Robot Pose	RobotPose	Blue	Estimated X,Y, and theta of the robot in the zone.
State Planner	MacroState	Pink	System behavior and macro canine movements.
Navigation	MotionCommand	Purple	Traverse and orientation commands for setting destination in navigation module.
Sensors	UWB	Black	X and Y UWB sensor measurements
Sensors	Accelerometer	Black-Striped	Theta accelerometer measurements.

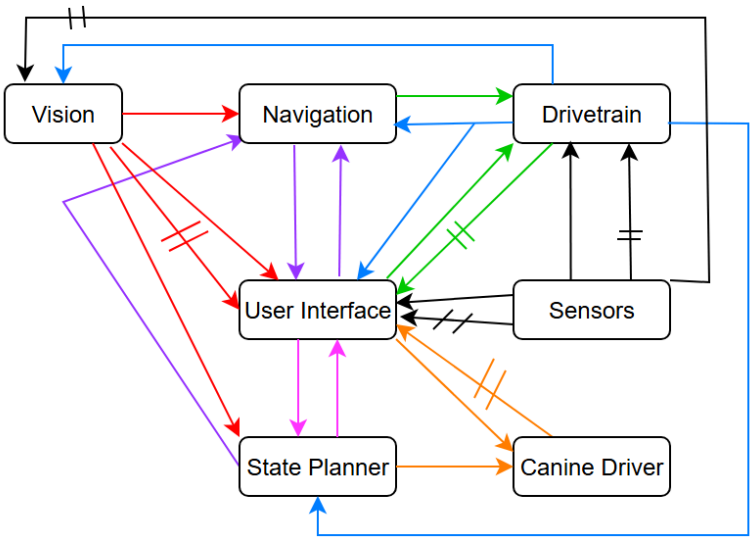


Figure 21. Schematic of inter-module connections following the protocol color scheme defined in Table 9. Modules are represented as rounded rectangles, with arrow directions indicating the publish-to-subscribe data flow between them.

The modules presented above were designed to decompose the system into focused, independently debuggable components. The Drivetrain and CanineDriver modules encapsulate the control logic interfacing with Team 18 and Team 19's hardware respectively. The Vision module handles object detection and pose estimation, while the Sensors module aggregates feedback from the UWB anchors and accelerometer. At a higher level, the StatePlanner module governs macro system behavior, coordinating chase, scan, and alert sequences by synchronizing commands across the CanineDriver and Drivetrain modules. The Navigation module is responsible for reference trajectory generation and obstacle-avoidant path planning. Finally, the UserInterface module serves as the primary debugging tool, providing remote actuation capabilities and system diagnostics to accelerate integration testing. The subsequent sections detail the functionality and implementation of each of these modules.

12.2 State Planner

The StatePlanner module manages high-level decision making across the robot system through three macro states. In the Scan state, the robot traverses the zone searching for turkeys. Upon detecting a turkey within the zone, the system transitions to the Chase state, triggering aggressive canine poses and audio while maneuvering toward the turkey's location. Once the Chase state times out, whether due to loss of detection or the turkey exiting the zone, the system enters the Alert state, which maintains a moderate deterrence presence. The Alert state serves as a transitional buffer: rather than immediately returning to passive scanning, the robot holds an intermediate posture that discourages turkeys from quickly reentering the zone. This is motivated by the risk of habituation, as a direct transition back to the Scan state may allow turkeys to

reenter before a new detection cycle begins, whereas the Alert state preserves deterrence while keeping operation within safe parameters.

To address scenarios involving multiple turkeys, the robot prioritizes the individual closest to its current position during chase sequences. To handle turkeys that disperse and linger within the zone, a ghost location routine is employed, whereby the Vision module records the last known position of any turkey that leaves the camera's field of view. The Scan state subsequently revisits these locations to recover and deter any remaining turkeys. A high level schematic of our three macro states is shown below in Figure 22.

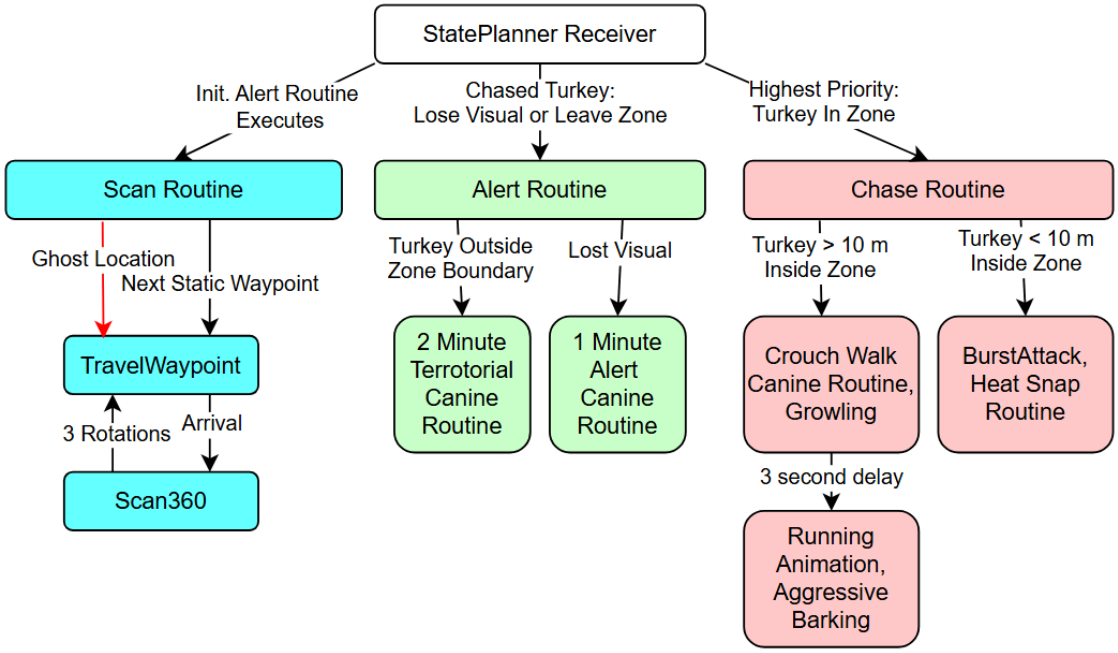


Figure 22. High-level StatePlanner flowchart with Scan, Alert, and Chase states color-coded in blue, green, and red respectively. Ghost locations, denoted in red, are assigned higher priority than the next static waypoint.

Due to the centralized role the StatePlanner plays within the system architecture, as evidenced by the numerous incoming and outgoing data channels shown in Figure 21, significant effort has been invested in the peripheral modules to simplify interfacing. This allows the StatePlanner to remain focused on macro-level state logic without being encumbered by lower-level implementation details. The resulting separation of concerns improves the debuggability of the StatePlanner and supports iterative refinement following field testing in simulated chase scenarios.

12.3 Computer Vision

The computer vision system is composed of three main components: computer vision model, camera detecting algorithm, and distance estimation algorithm. The design of these will be explained in detail by the following subsections:

12.3.1 Computer Vision Model

The computer vision model is responsible for detecting objects of interest in each frame captured by the camera. In this system, a YOLO26 object detection model is adopted due to its strong real-time performance, robustness, and compatibility with resource-constrained microprocessors.

The model takes an RGB image as input and outputs a set of detections, where each detection includes a bounding box, a class label (e.g., turkey or person), and a confidence score. To balance speed and accuracy, the model operates on resized inputs while preserving the original coordinate system for output mapping. This ensures that all detections remain consistent with the original image resolution.

The YOLO model is particularly suitable for this application because it performs object detection in a single forward pass, allowing the system to achieve near real-time performance. To improve generalization under varying environmental conditions—such as changes in lighting, background, and object pose—the training dataset is constructed from both publicly available online sources and on-campus collected data. These characteristics make the model an effective solution for robotic perception systems, where both efficiency and robustness are essential.

Camera Detection Algorithm

The camera detection algorithm is designed to improve the stability of detection results over time. Instead of relying on single-frame detections, the system applies a temporal filtering strategy to ensure that only consistent detections are considered valid. The principle is shown in Figure 23 below:

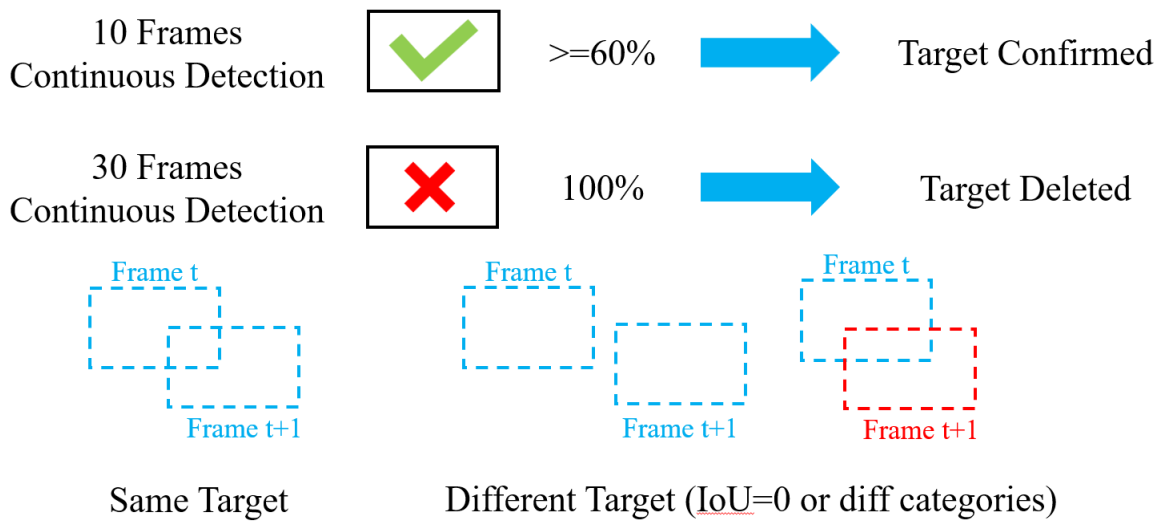


Figure 23. The Camera Detection Algorithm reference diagram. In the current setting, if a target is detected at least 6 frames of the previous 10 frames, we confirm that target and keep track of that until we lose the detection for a continuous 30 frames. Bounding boxes between frames are considered the same target if there is overlap and they are classified as the same categories.

The system maintains a short history of detections across consecutive frames. A target is only considered valid when it appears in a sufficient number of frames within a fixed window. This effectively removes transient false detections and reduces noise caused by environmental variations or model uncertainty.

To associate detections across frames, the system compares bounding boxes based on object class and spatial overlap. If two detections of the same class overlap, they are treated as the same target. Otherwise, a new candidate target is created. This simple association rule avoids the complexity of full multi-object tracking while remaining effective for scenarios with a limited number of targets.Although this approach may fail to distinguish between multiple targets that are spatially close or overlapping, such fine-grained differentiation is not required in this application. Since the primary objective of the system is directional awareness rather than precise multi-object identification, it is sufficient to treat nearby detections as a single target.

In addition, the system handles disappearing targets by tracking the number of consecutive frames in which a target is not detected. If a target remains undetected for a predefined number of frames, it is removed from the active target list. This mechanism prevents outdated information from persisting in the system while also providing robustness against temporary detection failures, such as brief occlusions or missed detections. As a result, the system avoids premature removal of valid targets while reducing the impact of sporadic false negatives during continuous tracking.

12.3.2 Distance Estimation Algorithm

The distance estimation algorithm utilizes a machine learning approach to predict the distance of targets and obstacles based on their bounding box produced by the computer vision model. While the horizontal angle of a target is determined by the center of its bounding box relative to the image center, the distance is calculated using a random forest predictive model.

To estimate distance, the algorithm extracts three key geometric features from the target's bounding box:

- Inverse Height: As an object moves further away, its vertical location in the image increases. By calculating the inverse of the bounding box location, the system creates a primary indicator of depth.

- Inverse Square Root of Area: Similarly, the overall pixel area of the bounding box shrinks as the distance increases. The algorithm normalizes this by taking the inverse square root of the area, providing a second robust metric for how far away the object is.
- Aspect Ratio: The ratio between the width and height of the bounding box helps the system account for the shape and orientation of the target, which refines the accuracy of the prediction.

These three features are fed into a Random Forest Regressor, a machine learning model designed to predict continuous values. During an initial training phase, the model analyzes a calibration dataset of bounding boxes that have been mapped to their actual, known distances. Through this training, the model learns the complex relationships between the size and shape of a bounding box in image-space and its physical distance in the real world.

During live operation shown in Figure 24, when the system detects a target, it calculates these geometric features in real-time and passes them through the trained model to instantly output an estimated distance.



Figure 24. The testing result for distance estimation algorithm. The model predicts sufficiently reliable distance and angle estimation which can help determine the position of the target in the map.

This algorithm provides a reliable and responsive estimation of distance for both our targets and obstacles. Based on our testing, the model can locate objects within a 1-meter range for distances under 5 meters, and within a 2-meter range for distances beyond 5 meters. Because our robot is not designed to approach the turkeys at an extremely close range, this level of data-driven accuracy is highly effective and more than sufficient for our prototype's operational needs.

12.4 Navigation

A significant component of the project involves navigating within the designated zone while avoiding both static and dynamic obstacles. Reliable navigation is essential for pursuing turkeys and maintaining full visual coverage of the zone. This section outlines the navigation stack, from receiving a target destination to generating a collision-free path, accounting for both static and dynamic obstacles. Path planning is performed using the A* algorithm, which finds the minimum-cost path to a given destination while avoiding obstacles. The resulting path is then processed by a trajectory tracker, which employs a Model Predictive Control (MPC) loop to generate feasible motion commands for the drivetrain module. This section also details the approach for mapping the zone offline to identify fixed obstacle locations and boundaries.

12.4.1 Zone Mapping

The first step in developing and implementing our navigation system is creating a map of our designated zone. This will allow us to track our movements within the zone, and our relative position to other objects or obstacles. To start, four UWB anchors will be placed, one at each corner of the defined area. The subsequent zone will be subdivided into 1 by 1 meter squares, as seen in Figure 25 below, which we will further detail in the following sections.

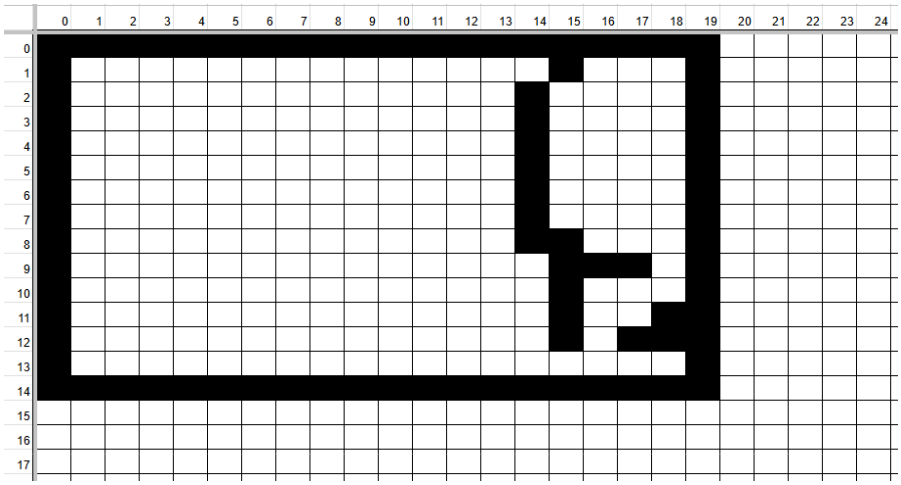


Figure 25. CSV file map representation with black tiles as obstacles or boundaries and white tiles as vacant and traversable space.

Aside from dynamic objects (humans, turkeys etc.) traversing the zone, there will also be static obstacles that need to be represented in our map beforehand. To do this, we will use our UWB tag, and walk to each of the static objects that we want to avoid. The preset UWB anchors will allow us to calculate the obstacle position in the zone, and mark it in our zone representation. Note that, the top left anchor will be used as our XY datum, which we will use to represent the relative positions of each object.

Within our representation of squares, key squares are selected as our scanning waypoints. The navigation algorithm will tell our robot to traverse between these preset locations, and take time to scan, ensuring thorough coverage of the zone.

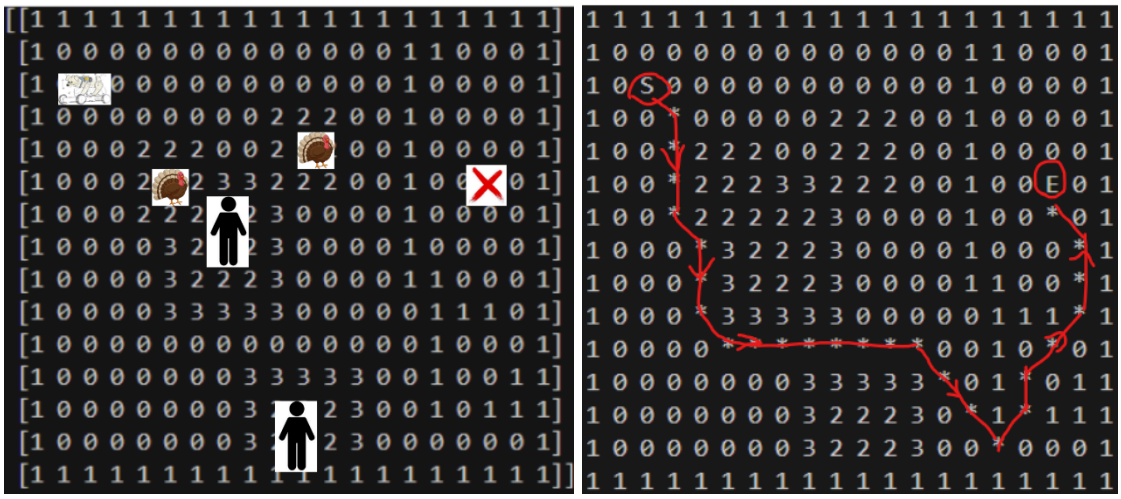
12.4.2 Zone and Obstacle Representation

Once the zone map has been constructed with fixed obstacles identified, it is imported via a CSV file and read into a Python script using standard parsing libraries. The map is then used to construct a 2D grid array using NumPy, a Python library designed for scientific computing and matrix operations. Within this grid, static obstacles are denoted by a value of 1 and traversable locations by a value of 0. Dynamic obstacles are represented by a value of 2, with their positions determined and placed based on VisionObject data received from the Vision module. Dynamic obstacle entries are continuously updated and cleared in response to Vision module commands, while static obstacle entries remain fixed for the duration of operation. Values greater than 2 represent custom traversal costs, penalizing movement through undesirable terrain without prohibiting it entirely. Such regions may include mud, wood chips, or other areas that are preferable to avoid but permissible to traverse if necessary.

To prevent direct collisions with both personnel and turkeys, detected objects are enclosed within a buffer region of obstacle tiles to provide additional clearance. Turkeys are currently represented as a 3x3 grid of obstacle values, while personnel are represented as a 3x3 grid enclosed by an additional ring of penalized traversal weights. These configurations are subject to refinement through testing, but provide an initial degree of robustness against uncertainties in the vision-based object detection and localization estimates.

12.4.3 Path Generation

Given the continuously updated dynamic map, a preplanned trajectory, updated at a 2 Hz frequency, is generated upon receipt of a zone traversal command to reach the desired destination. Path planning is performed using the A* algorithm, a heuristic-based search approach that efficiently identifies a low-cost route through the grid while avoiding both static and dynamic obstacles. A simulated example is presented in Figure 26, in which personnel and turkeys have been arbitrarily placed within the zone map from Figure 25, with the current and target robot positions marked. This example demonstrates the capability of the A* path generation algorithm to produce collision-free routes under representative operating conditions.



(A) (B)

Figure 26. (A) The zone map with turkeys, personnel, and the robot's start and destination positions denoted by the dog icon and X marker, respectively. (B) The planned route generated by the A* algorithm, outlined in red, to reach the target destination.

Given the padding applied to both personnel and turkey representations in the zone map, the A* algorithm will be unable to find a path that reaches the exact center of a target object, as these cells are marked as obstacles. Consequently, the algorithm must exhaustively search all reachable grid tiles before terminating, which is anticipated to introduce performance limitations. To mitigate this, heuristic pruning strategies have been incorporated to accelerate pathfinding. These include early termination upon reaching a preset proximity threshold to the destination, and pruning of candidate trajectories that deviate beyond a preset distance from the straight-line path to the target. Although these modifications depart from the standard A* formulation, they are appropriate given the anticipated operating conditions, where obstacles are sparsely distributed and the algorithm is primarily employed to avoid moving obstacles rather than navigate complex corridors or maze geometries that would demand a more exhaustive search.

12.4.4 Trajectory Tracking

Once a path has been generated, it must be converted into a sequence of feasible control inputs to be passed to the drivetrain module. This is accomplished through a nonlinear Model Predictive Control (MPC) trajectory tracking formulation. MPC operates as a receding horizon optimization procedure, computing control inputs that best align the robot with the planned trajectory while satisfying a set of defined constraints. State constraints enforce obstacle avoidance with respect to both static and dynamic obstacles, while input constraints enforce maximum allowable traversal speeds. Additional penalty terms can be introduced to shape the optimization behavior, including penalizing rapid changes in control inputs between timesteps and independently weighting deviations in X, Y, and Theta from the reference trajectory.

The MPC algorithm will be implemented using the CasADi library, which provides Python integration of the mathematical framework required for nonlinear optimization. Additional research and development are required to complete this implementation. Should this approach prove too complex to integrate within the remaining project timeline, a PD controller will be adopted as an alternative, providing a feedback-only trajectory tracking strategy based on the error between the current and desired positions along the planned path.

12.5 Drivetrain Actuation

The subteam is responsible for developing the software to control Team 18's drivetrain, which employs a skid-steer configuration consisting of four brushed DC motors, two on each side, with steering achieved by differentiating left and right motor speeds. To accommodate the current draw requirements of the selected motors, the *RoboClaw Solo 60A Motor Controller* was selected for integration. This device offers extensive protocol support, including motor current monitoring, encoder feedback, speed control, and built-in PID configuration, as well as an offline motor configuration interface called *MotionStudio*. Four controllers are to be utilized, one per motor, operating in packet serial mode in accordance with the RoboClaw packet serial protocol. Although an existing Python interface was provided by RoboClaw, it was deemed unsuitable for the team's requirements. A minimal interface was instead developed, exposing only the subset of commands directly pertaining to the required functionality rather than the exhaustive set provided by the manufacturer. The rest of this section outlines our plans for motor control, PID gain selection, and safety routines on a firmware level.

12.5.1 Motor Control

Due to the comprehensive protocol provided by the manufacturer, controlling the DC motors involves limited software complexity. Four USB-A to USB Micro-B connections interface the Raspberry Pi with each of the four motor controllers. Motor speeds are commanded over serial using the *Drive M1 With Signed Speed* command, while encoder readings are retrieved using the *Read Encoder Speed M1* command. A schematic of the interfacing routing is presented in Figure 27 below.

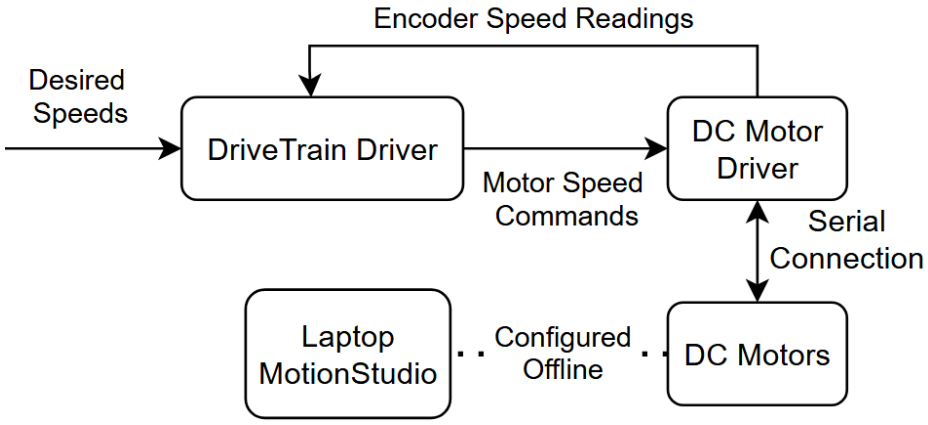


Figure 27. Schematic of drivetrain motors interfacing. The drivetrain driver sends motor speeds and requests speed readings from the Motor Controller. Numerous motor settings can be configured offline using MotionStudio, indicated with the dotted connection.

12.5.2 PID Gain Tuning

The motor controllers provide built-in PID control, enabling closed-loop speed regulation from external encoder readings. The current tuning strategy involves collaborating with Team 18 to obtain a motor model transfer function relating speed reference inputs to corresponding speed outputs. Once derived, this model will be imported into MATLAB, where the *Control System Toolbox* will be used to perform gain tuning targeting reduced rise time and overshoot. Tuning will be conducted primarily through iterative adjustment within MATLAB's *Control System Designer*. Time permitting following full drivetrain assembly, manual gain iteration may also be performed based on visual inspection, with MotionStudio's built-in signal tracker used for offline debugging.

12.5.3 Firmware Safety Routines

Rudimentary motor safety routines have been implemented to mitigate the risk of overcurrent conditions resulting from motor stalling or significant wheel slip. To prevent overcurrenting, motor current is sampled at a rate of 10 Hz. If current readings continuously exceed a threshold of 40 Amps over five consecutive samples, the commanded voltage is automatically set to zero, providing a software-level safeguard against motor damage. In addition, the motor controllers provide a hardware-level maximum current limit, which will shut off the motors if current reaches a configured ceiling. Both strategies are employed in concert to ensure motor current remains within the safe operating limits of the electronics and wiring.

Slip detection is achieved by comparing encoder-measured motor speeds against positional estimates derived from Ultra-Wideband (UWB) sensor readings, which provide relatively accurate zone-level coordinate measurements. From these two sources, a slip ratio is computed

using Equation 1, where Venc denotes the velocity measured by the wheel encoders and Vuwb denotes the velocity estimated from the UWB anchors.

$$Slip\ ratio\ =\ abs(\frac{Venc - Vuwb}{Venc}) \quad (1)$$

Drawing from established literature, slip ratios below 0.4 are considered adequate for maintaining traction in soil-based environments [21]. Accordingly, a three-tier slip classification scheme was adopted: no slip for ratios below 0.1, slip warning for ratios between 0.1 and 0.3, and slip error for ratios between 0.3 and 0.5. In response to each classification, motor speed commands are scaled by a factor of 1.0, 0.5, and 0.0, respectively, with corresponding status indicators reported in the user interface. Further collaboration with Team 18 is required to refine the slip response mechanism, and operator acknowledgment may be introduced for severe slip conditions. The effectiveness of the heuristic anti-slip model will be evaluated through testing, with adjustments made to its complexity and parameters as necessary.

12.6 Localization

Accurate estimation of the robot's pose, defined by position coordinates x and y and heading angle theta, is critical for reliable navigation and turkey pursuit within the designated zone. Insufficient localization accuracy would cause dependent navigation and object detection algorithms to degrade rapidly. To address this, an Unscented Kalman Filter (UKF) approach was adopted as a robust localization solution. In all Kalman Filter formulations, estimation proceeds through two stages: a Predict step and an Update step [22]. The Predict step propagates the current pose estimate forward using a differential drive motion model, with positional uncertainty growing over time as errors accumulate in the absence of an external reference. The Update step incorporates sensor measurements of x, y, and theta to reduce this uncertainty by reconciling the predicted estimate with observed data. The Unscented Kalman Filter variant is specifically designed for nonlinear systems, a criterion satisfied by the geometric nonlinearity introduced by the heading angle theta in the system dynamics [23]. The following sections detail the implementations of the Predict and Update steps.

12.6.1 Predict Step

Encoder readings from the motor controllers are used to predict the robot's updated pose based on measured wheel speeds. The high resolution of the encoder measurements enables precise state estimation within the zone. The underlying dynamics governing this propagation are presented in Equation 2, where x, y, and theta represent the position and heading of the robot, v_r and v_l denote the right and left wheel speeds respectively, L is the wheelbase, and delta t is the sampling interval. The y direction is inverted to match the positive downward coordinate system defined by the zone map.

$$x(k+1) = x(k) + \frac{v_r + v_l}{2}\cos(\theta(k)) \cdot \Delta t \quad (2A)$$

$$y(k+1) = y(k) - \frac{v_r + v_l}{2}\sin(\theta(k)) \cdot \Delta t \tag{2B}$$

$$\theta(k+1) = \theta(k) + \frac{v_r - v_l}{L} \cdot \Delta t \tag{2C}$$

Following the UKF algorithm, a set of sigma points is distributed around the current pose estimate and propagated through the nonlinear differential drive motion model. Each sigma point represents a locally plausible robot pose, which are recombined using a set of predefined weights to produce an updated mean pose and covariance matrix. To account for unmodeled dynamics and actuator noise, the process noise covariance matrix is added to the predicted covariance at each iteration. By the nature of the algorithm, each Predict step causes uncertainty to grow as covariance accumulates in the absence of a corrective measurement.

12.6.2 Update Step

The Update step incorporates two sources of sensor measurements. The first is a planar position estimate of the robot's x and y zone coordinates derived from Ultra-Wideband (UWB) anchors, which measure distances between anchor points and triangulate the robot's position relative to known reference locations using regression to minimize positional error. The second is a heading estimate provided by a 9-DOF inertial measurement unit, which leverages the Earth's magnetic field to approximate the robot's angular orientation in the horizontal plane. Both measurements are used to continuously update the UKF, reducing covariance uncertainty with each sensor reading as they validate the current pose estimate. Tuning the relative covariance gains between the Predict step and the UWB and accelerometer Update steps will be necessary to appropriately weight each measurement source according to its reliability. This process will require iterative field testing to optimize overall estimation performance.

12.7 Canine Actuation

Within the defined scope, the control logic for Team 19's robotic canine encompasses servo motor movements for the four legs and head, as well as audio commands delivered through a Bluetooth speaker. To coordinate development efforts, templated specifications were constructed and subsequently completed by Team 19, serving as the implementation basis for the software architecture. This section is organized into two primary elements: servo motor control and audio control. A schematic of the current software design is presented in Figure 28, detailing the command routing from the CanineProtocol to actuated behavior.

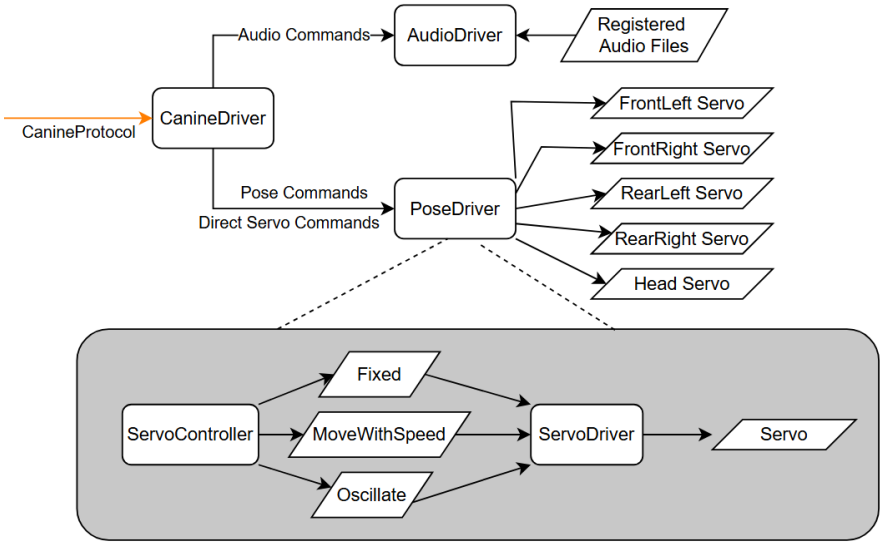


Figure 28. High-level software schematic of the canine driver. Python modules are represented as rounded rectangles, with arrows denoting communication paths between them.

12.7.1 Servo Control

Team 19 specified requirements for high-level canine states that emulate behavior during pursuit and alert conditions. To implement these states in a manner that is both responsive to updates and practical for motor tuning, a three-level control hierarchy was developed to translate macro behaviors into servo commands. At the lowest level are Angle commands, which directly set individual motor positions. Above this are MotorStates, which define movement profiles such as oscillatory motion between two set points. At the highest level are Poses, which map desired canine behaviors into coordinated motion profiles across all five motors. As illustrated in Figure X, Poses are sent to the PoseDriver, where they are decomposed into MotorStates by the ServoController and subsequently forwarded to the ServoDriver as Angle commands to drive the servos.

Inside the ServoDriver script, Angle commands represent the lowest level of the software hierarchy and directly interface with the servo motors. Control signals are generated using software-based PWM at 50 Hz and applied to the servo control pins. Each angle command is converted to a pulse width via linear interpolation between the minimum and maximum angle–pulse width mappings defined in the servo specifications. To satisfy Team 19's constraints, commanded angles are bounded such that leg motors remain within −45° to 45° and the head motor within −60° to 60°. Any command exceeding these limits is saturated to the nearest allowable value.

Processed in our ServoController script, MotorStates are collections of motion profiles assigned to individual servo motors, mapping one-to-one with each motor. Together, they define Team

19's motor requirements for a given pose. Each MotorState follows one of three primary motion profiles: Fixed, which commands the motor to a specified angle and holds that position; Oscillate, which transitions the motor between two set points at a defined period; and Random, which selects an angle within a specified range at a defined period.

Poses are macro-level movements of the canine, encompassing walking and running animations as well as various static positions. The set of supported poses was derived directly from Team 19's requirements, as communicated through the project specifications (see Appendix X for Team 19's full specifications). Poses serve as the interface level to the canine driver, implemented within the PoseDriver module, which executes commands corresponding to the requested pose. Upon execution, a pose updates the system's five servo motor MotorStates, reassigning their motion profiles to match the desired movement behavior. To prevent transient motor angles from inducing oscillation, poses are assigned a default wait period of 0.5 seconds, though this value is subject to change following testing.

The three-level approach of Angle, MotorStates, and Poses was designed to manage the complexity arising from the real-time demands of the control system. Each layer communicates with the layer below through asynchronous channels, enabling a clean separation of concerns. Safety specifications and angle operations are encapsulated within the Angle level, such that any motor changes are localized to that layer without propagating to upper levels. Similarly, new motion profiles can be introduced at the MotorStates level without disrupting existing profiles, and poses can be added or removed at the Poses level as required by Team 19.

12.7.2 Audio

Audio is a central component of the robotic canine's fear response, replicating hunting-related auditory behaviors such as barking and growling. Per Team 19's request, four audio signals are implemented, as summarized in Table 10.

Table 10. The four audios and respective descriptions provided by Team 19.

Number	Name	Description
1	BarkShort	Single short bark used as a warning or attention cue
2	BarkBurst	Rapid bark sequence (2–3 barks) used during active chase
3	Growl	Low growl indicating territorial warning
4	AggressiveBark	Continuous barking for 2–3 seconds is used during chase mode

Audio playback is implemented using *pygame*, a Python library commonly used in game development. It provides a simple interface and supports asynchronous control of sound

playback. To maintain predictable behavior, only one audio signal may play at a time under a last-request priority scheme; any currently playing sound is immediately terminated when a new command is issued. A helper “Stop” command is also supported to terminate all active audio for debugging purposes.

12.8 User Interface

Given the numerous actuators, states, and configurations involved, integration is anticipated to be a significant challenge. To facilitate final integration and testing, a User Interface was developed to provide remote access to four primary components. The top left panel displays a live camera feed with active object classifications. The top right panel presents the zone map, indicating fixed obstacles, waypoints, and the robot's current position. The bottom right panel provides manual control inputs for servo actuation, drivetrain speed commands, and audio playback. The bottom left panel displays system diagnostics, including current pose estimates, motor speeds, and the active macro state. A screenshot of the User Interface is presented in Figure 29 below.

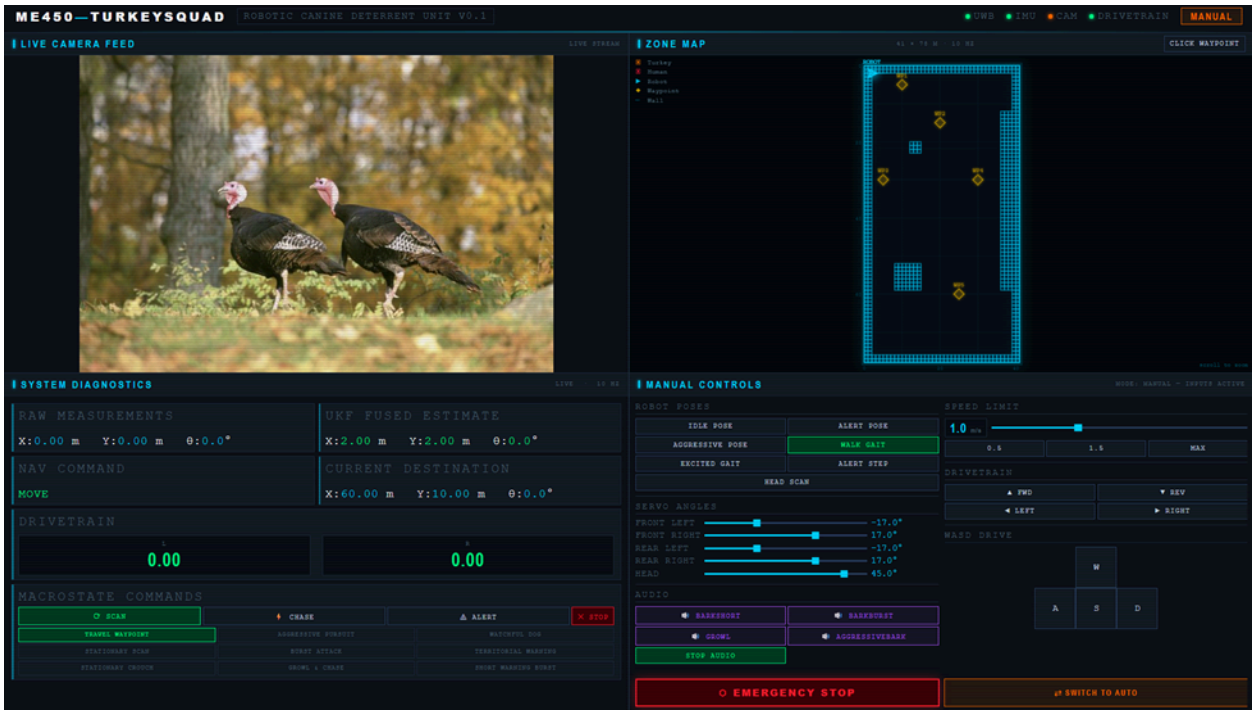


Figure 29. Screenshot of the User Interface, consisting of a live camera feed, zone map, manual control panel, and system diagnostics panel.

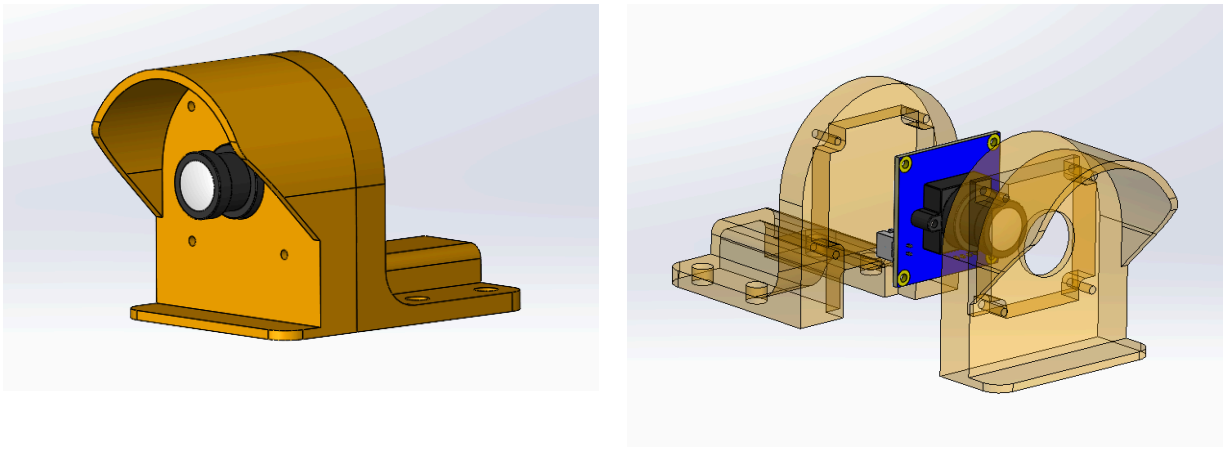
The User Interface is designed to operate remotely over an ESP32-hosted wireless network, with the frontend implemented in HTML, CSS, and JavaScript, while the backend developed in Python using FastAPI. This architecture provides real-time system diagnostics and enables field testing and debugging within the constraints of the semester timeline. The User Interface is also

intended to be showcased at the final project expo, where it is expected to serve as a compelling demonstration of the capabilities of our final deliverable.

12.9 Mechanical Assemblies

Our subteam chose to take on the responsibility of designing and fabricating the camera mount and UWB housing units that would be utilized for our final prototype. This choice allows us to quickly iterate features of the camera mount according to our testing, and similarly adjust the UWB housing units depending on our needs.

To design the camera mount, we had already determined the location on the front of the drivetrain, so the main factors to consider were stability, robustness in outdoor environments, and the vantage point that we would be capturing images from. The final camera mount is seen in Figure 30 below.



a. **b.**

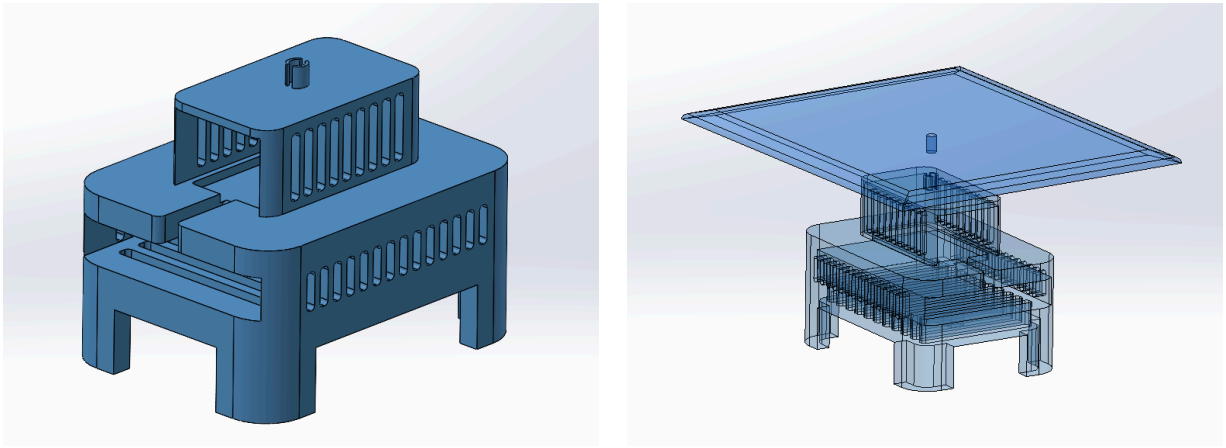
Figure 30a, b. CAD of the camera mount, and the separated assembly components, respectively.

To eliminate vibrations and maximize stability, the mount was designed in two parts. The rear component has 4 mounting holes that constrain it to the drivetrain. The camera is then pressed between the front and rear mounts, and held in place by 4 screws to ensure that vibrations do not impact our ability to detect objects. A lens hood was implemented to protect the camera lens from dirt, snow and moisture, as well as stray light that could enter the lens. Finally, a rear cavity and tunnel were included, giving space for the camera's wiring and USB cable to fit into.

For our material, we chose to use PETG and 3D print the mount components. We wanted to allow for fast iteration depending on the effectiveness of our vantage point during testing, which additive manufacturing will allow for. This also gives us the added benefit of light weight, in comparison to other materials like aluminum or steel. While alternatives like PLA were

considered, the potentially long operation in the outdoors meant the possibility of deformation from prolonged exposure to sunlight as well as precipitation, which was a concern. PETG is a far more weather resistant material and also has a higher glass transition temperature, which is the reason for our selection.

When designing our UWB anchor housing, the primary functions we needed to consider were stability and outdoor environment robustness. An image of our UWB anchor housing CAD can be seen in Figure 31 below.



a. **b.**

Figure 31a, b. CAD of the UWB housing unit, and the attachable roof assembly, respectively.

The design consists of two separate housing spaces for both the UWB microcontroller as well as the corresponding power bank. Cooling grills are implemented to allow for heat dissipation, and an attachable roof was designed to prevent precipitation from entering the housing. PETG was selected as the printing material with the same reasoning as the camera mount.

12.10 Wiring

A list was created for every wire across the three subteams. For each wire, the connection type, components connected, length, color, gauge, voltage, max current, and shielding are determined. Visual of the full wiring diagram is shown in Figure 32 below.

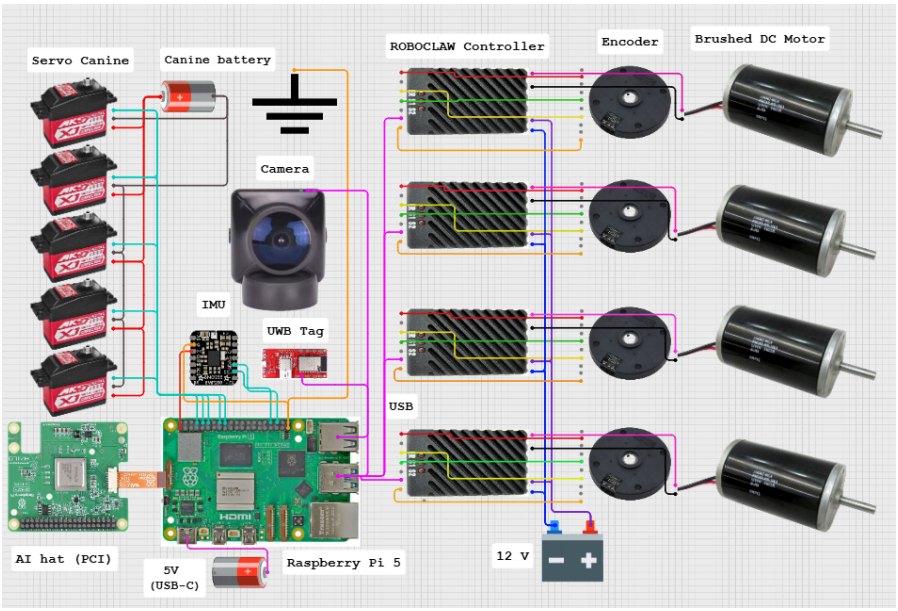


Figure 32. Wiring diagram for the entire robot across three subsystems of canine, drivetrain, and vision & navigation.

Due to the different responsibilities of each subteam, we determined that the best approach was to assign specific sections of the wiring scheme to each. The following diagram in Figure 33 below shows the specific elements that belong to our team (17).

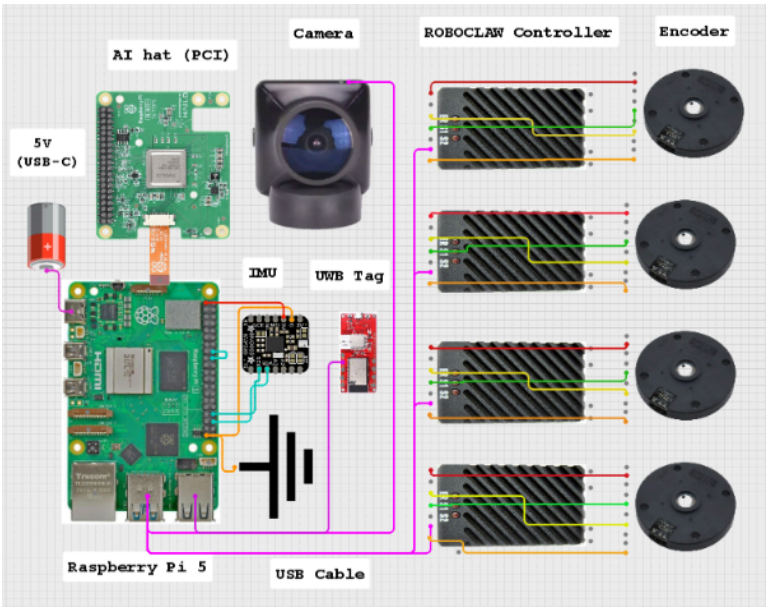


Figure 33. Wiring diagram for all wires under responsibility of team 17.

13 ENGINEERING ANALYSIS

This section evaluates the computer vision and actuator control modules to confirm the system performs reliably in real world conditions. The analysis focuses on accuracy, safety margins, and responsiveness to validate the transition from theoretical design to integrated deployment.

13.1 Computer Vision

To systematically assess the effectiveness, robustness, and real-world applicability of the proposed computer vision system, a multi-level evaluation framework is developed, encompassing model-level accuracy analysis, system-level continuous performance assessment, and real-world field validation.

13.1.1 Model-Level Evaluation

At the model level, the performance of the YOLO26 detector is evaluated using a labeled test dataset consisting of individual images. To ensure a fair and unbiased evaluation, the test dataset is constructed from newly recorded data that is entirely separate from the training dataset, rather than being randomly split from the original dataset.This design choice is critical because a portion of the training data is collected from video sequences, where consecutive frames are highly similar. A random split could therefore result in nearly identical frames appearing in both the training and test sets, leading to overly optimistic performance estimates due to data leakage. By using a completely independent dataset, the evaluation more accurately reflects the model's ability to generalize to unseen scenarios.

A confusion matrix shown in Figure 34 is used to quantify classification performance across different object categories (e.g., turkey, person, and background). From this, key metrics such as precision, recall, and overall accuracy are computed. These metrics provide insight into the model's ability to correctly detect targets while minimizing false positives and false negatives.

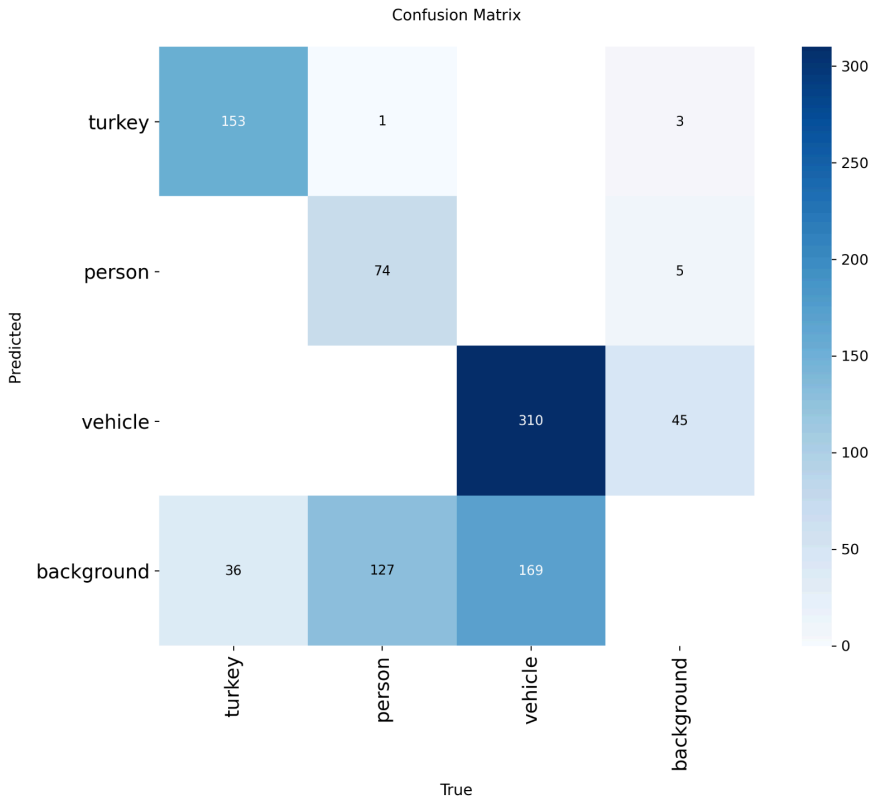


Figure 34. An example of a confusion matrix for the computer vision model. We expect high precision (prevent error detection) while still try to preserve a certain level of recall (prevent missed detection).

13.1.2 System-Level Evaluation

At the system level, the performance of the complete detection pipeline, including temporal filtering, target association, and distance estimation, is evaluated using continuous video input. Unlike model-level evaluation, which focuses on per-frame accuracy, this stage assesses how the system performs over time under realistic operating conditions.

A key aspect of this evaluation is the maximum detection distance, defined as the farthest distance at which a target can be reliably detected and consistently maintained across consecutive frames. This metric reflects the practical sensing range of the system and is particularly important for downstream navigation decisions. In addition, the stability of the processing frame rate is evaluated to ensure that the system can maintain near real-time performance without significant fluctuations that could impact responsiveness.

The evaluation of the distance estimation algorithm is conducted with a focus on its conservative behavior rather than absolute accuracy. The system is intentionally designed to underestimate the true distance to a target, ensuring that the robot reacts safely by assuming objects are closer than they actually are. Therefore, instead of minimizing absolute error, the evaluation emphasizes the

consistency and predictability of this conservative bias. Specifically, the analysis examines whether the estimated distance remains reliably below the ground-truth distance and whether the magnitude of underestimation follows a stable and monotonic trend as the true distance increases. This approach ensures that the distance estimation is both safe and sufficiently informative for decision-making, even if it is not metrically precise.

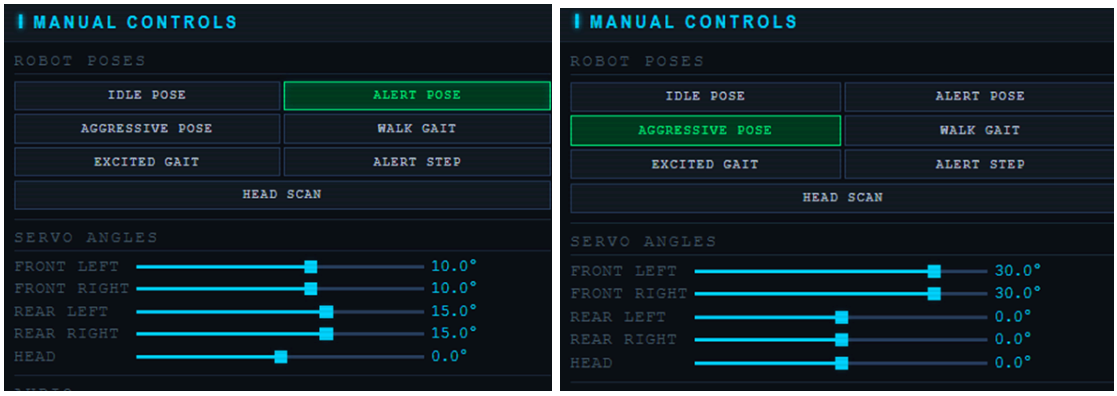
13.1.3 Field Testing

To validate the system under practical operating conditions, field testing is conducted in real-world environments. Unlike controlled dataset or laboratory evaluations, this stage focuses on the robustness and reliability of the system when exposed to uncontrolled factors such as varying lighting conditions, complex backgrounds, and dynamic scenes. These tests are essential for assessing whether the system can maintain stable performance outside of idealized settings.

The evaluation examines the system's ability to consistently detect and maintain targets in the presence of multiple objects, as well as its robustness to partial occlusion and temporary detection loss. Particular attention is given to tracking stability over time, ensuring that detections do not fluctuate excessively or disappear unpredictably. Through these tests, the system's overall reliability and adaptability to real-world scenarios are assessed, providing a final validation of its readiness for deployment.

13.2 Actuator Control

To assess the correctness and latency of actuator interfacing, the User Interface serves as the primary verification tool during development. By issuing commands through the interface and observing the resulting behavior, discrepancies can be identified and corrected prior to field testing. A screenshot of the User Interface is presented in Figure 35, illustrating two canine poses with their corresponding servo angles defined in accordance with Team 19's specifications.



(A) (B)

Figure 35. User Interface screenshots highlighting the canine servo actuation panel. (A) Alert pose and (B) Aggressive pose, with corresponding servo angles defined in accordance with Team 19's specifications.

14. PROBLEM ANALYSIS AND ITERATION

To ensure the final system meets or exceeds the requirements outlined in Table 2, we will employ extensive empirical testing and rapid prototyping alongside design development. Purchased hardware will be evaluated using dedicated test setups that include performance and reliability assessments. Custom software modules will be validated and iterated at the submodule level prior to full system integration. This is intended to significantly simplify software debugging and accelerate integration with the mechanical assemblies developed by Team 18 and Team 19.

Theoretical analysis will be used primarily to inform the selection of camera hardware and control algorithm tuning. The following sections describe existing and planned measures to verify that the design satisfies all requirements before final system integration.

14.1 Computer Vision

During the development of the computer vision system, several challenges were identified through testing and iterative refinement. These issues primarily relate to model performance at higher resolutions and the robustness of the distance estimation algorithm.

14.1.1 High-Resolution Model Performance Degradation and Dataset Scaling

While higher-resolution images theoretically provide more detailed visual information for detecting distant targets, initial experiments showed that the model trained on higher-resolution inputs performed worse on the test dataset compared to a lower-resolution baseline. This degradation is attributed to insufficient dataset diversity and scale, which limits the model's ability to generalize in a higher-dimensional input space. To address this issue, the dataset size was significantly expanded by incorporating additional training samples from both online sources and newly collected data. This increase in data diversity improved the model's generalization capability and restored the expected performance gains from higher-resolution inputs.

14.1.2 Pose Sensitivity in Distance Estimation and Data-Driven Improvement

Since the estimation relies heavily on bounding box geometry, changes in object orientation or partial occlusion can lead to significant fluctuations in the predicted distance. To mitigate this limitation, an improved approach is proposed in which multiple geometric features of the bounding box—such as width, height, aspect ratio, and area—are extracted and used as input features. By collecting ground-truth distance data through controlled camera experiments, a lightweight regression model (e.g., a random forest decision tree) can be trained to learn the mapping between these features and actual distance. This data-driven approach is expected to reduce sensitivity to pose variation and improve the robustness and consistency of distance estimation.

14.2 Controls

To meet the Movement requirements, the task is divided into two categories: Navigation, responsible for high-level decision-making and path planning, and Control Actuation, which interfaces with the motors and executes command signals. Preliminary navigation algorithms are developed primarily through empirical testing, whereas control actuation and gain tuning leverage dynamic motor models and control theory to determine key performance metrics such as steady-state behavior and rise time.

14.2.1 Control Actuation

The low-level motor control will use PID algorithms, with gains tuned using a preliminary linear motor model relating input voltage to translational speed. Encoder measurements will provide estimates of actual speed, and the PWM input will be adjusted to match the reference velocity.

Gain tuning will be performed in MATLAB using the Control System Toolbox, evaluating performance metrics such as overshoot, rise time, and settling time. While no strict requirements exist for these metrics, the system will be tuned to satisfy macro-level speed and deceleration specifications.

Once Team 18 finalizes motor selection, benchtop testing will be conducted with the motor drivers and software control. Tests will include scenarios such as rapid acceleration, steady-state travel, and turning. Control performance and any hardware-related issues identified during testing will inform software adjustments to optimize responsiveness and power efficiency.

14.2.2 Navigation

The navigation algorithm will be preliminarily validated using a custom Python-based simulation tool that models the robot's dynamics as a differential drive system. Validation will involve analyzing the resulting behavior and trajectory performance, with initial iterations focused on optimizing obstacle avoidance and minimizing travel time. Insights from these simulations will guide tuning of both the A* and MPC algorithms. Additionally, computational performance will inform adjustments such as selecting appropriate MPC time horizons and implementing pruning heuristics in A* to reduce decision times.

Once the physical prototype is assembled and ready for validation, field testing will be conducted by navigating the zone and executing the Chase Protocol on a realistic turkey decoy. Evaluation will be primarily observational, supplemented by the user interface, which will facilitate testing of individual submodules and core system functions.

15 PROBLEM DOMAIN AND ANALYSIS

Current progress is defined by the technical challenge of balancing perception accuracy, specifically the stable detection of turkeys within a 15-meter range, against the computational latency of embedded hardware. This analysis evaluates the domain gaps in our hybrid dataset, the hardware-software integration required to ensure electrical and navigational stability, and the optimization of real-time pathfinding algorithms. By addressing these core constraints, the team aims to establish a robust framework for high-level intelligence capable of navigating dynamic campus environments while maintaining strict safety buffers around pedestrians and vehicles.

15.1 Computer Vision

Operating the robot in a real-world campus environment requires collecting representative on-campus data so the model can learn the visual characteristics of our deployment zone. Specifically, we need photos and videos of wild turkeys moving through campus spaces. Despite coordinated efforts from three subteams, the inherently random and infrequent appearance of turkeys has limited our dataset size. In addition, manually labeling hundreds of images with bounding boxes is time-intensive and remains a major bottleneck in expanding our real-world dataset. To address this, we are considering organizing a public data collection initiative to gather more on-campus footage.

To mitigate overfitting and maintain sufficient training diversity, we have supplemented our dataset with publicly available online data. The mature computer vision ecosystem provides abundant urban scene datasets, such as street images and pedestrian datasets, that support our broader operational requirements. However, datasets featuring wild turkeys in urban environments are extremely limited. Most available turkey datasets depict birds in rural or natural habitats, which differ significantly from campus settings and can reduce recall when deployed in our target environment. Therefore, our on-campus dataset is essential to compensate for this domain gap. Determining the optimal balance between online datasets and locally collected campus data is a key challenge that requires extensive model testing and validation.

Another major challenge is the tradeoff between computational constraints and image resolution. Our system must reliably detect turkeys within a 15-meter range, which requires sufficient image resolution to ensure stable detection performance. For YOLO-based models, maintaining a minimum object pixel height of approximately 20 pixels is generally desirable for robust detection. However, increasing image resolution significantly increases computational load and reduces inference speed on our embedded hardware. One potential solution is to offload inference to an external Neural Processing Unit (NPU), but this approach requires further benchmarking and integration testing to determine feasibility within our system constraints.

15.2 Controls

The successful deployment of our robot requires integration between the software control logic and the physical hardware provided by Team 18 and Team 19. Our primary responsibility

involves developing the core control software and selecting the electrical hardware necessary to interface with the robot’s drive systems. A critical technical challenge in this integration is ensuring power voltage agreement across all sub-modules. Inconsistent voltage levels between the microcontroller, sensors, and the chassis power supply could lead to hardware instability or damage, necessitating rigorous electrical validation before full-scale testing.

Navigation software complexity remains a significant challenge, particularly regarding the accuracy of the robot's estimated pose. Precise localization is the foundation of effective movement; any drift in pose estimation will result in cumulative pathing errors. Furthermore, the environment is highly dynamic, requiring the navigation algorithm to account for the unpredictable movements of both pedestrians and turkeys. To ensure comprehensive coverage of the operation zone, we are developing a systematic protocol for scanning the full zone, allowing the robot to patrol efficiently while maintaining high detection confidence across the entire operational space.

To address latency concerns, we are focusing on the optimization of A* and Model Predictive Control (MPC) algorithms. These pathfinding and control loops are computationally expensive and can introduce lag that compromises the robot's real-time responsiveness. To mitigate this, we are designing the system with asynchronous behavior, allowing the navigation and control tasks to operate on separate threads. This ensures that a delay in path recalculation does not freeze the robot's low-level motor commands, maintaining smoother and safer motion.

Finally, operational reliability is threatened by limited Wi-Fi access within the target zone, which impacts the Controller User Interface (UI). Since the UI relies on a stable network connection for remote monitoring and manual override, we must develop a robust fail-safe mechanism. We might first try to use a hotspot on our phone. If the connection remains unstable, a portable Wi-Fi signal generator may be needed.

16 VERIFICATION AND VALIDATION

Our team is using a Traceability Matrix to link every requirement to a specific test method and plain language validation question. This approach ensures no critical performance or safety metric is overlooked during the transition from integration to field testing. Some initial software and control logic was verified with a prototype drivetrain, and final validation was performed through field testing on North Campus. Table 11 below summarizes the V&V strategies for each requirement as defined in the master traceability log.

Table 11. Team 17 simplified Verification and Validation traceability matrix

ID	Requirement Name	Specification Summary	Verification Method	Validation Question
Environment Detection				
1.1.1	Turkey Detection Accuracy	> 90% frames classified correctly; < 5% positional error.	Generate a confusion matrix using field and online datasets.	How well can the vision system detect the turkey?
1.1.2	Bounding Box Accuracy	Intersection over Union (IoU) > 0.7 for humans at 1–5m.	Compare human test data to manually labeled ground-truth boxes.	How well can the vision system detect humans?
1.2.1	Zone Scan Time	360° coverage of the entire zone within 120 seconds.	Time the autonomous traversal/rotation with a stopwatch.	Can the system view the whole zone fast enough?
1.2.2	Max Latency	Classification latency < 1000 ms per image.	Verify prediction times across a large dataset.	Can the vision model run fast enough?
1.2.3	Computing Resources	< 50% CPU usage and < 70% RAM consumption.	Reference hardware specs and monitor active usage on the Pi 5	Will the hardware stay within its recommended limits?
Movement				
2.1.1	Zone Confinement	Min 2m buffer from boundaries; 4m from southern road.	Test navigation software with the prototype drivetrain.	Can the system maintain a safe boundary?
2.1.2	Obstacle Avoidance	Speed < 0.5 m/s within 4m of humans; 1m min clearance.	Calibrate software commands against wheel/motor rotation rates.	Will the system account for pedestrian proximity?

2.1.3	Max Travel Speed	< 3 m/s (pursuit); < 2 m/s (neutral).	Manually trigger pursuit sequences and record speeds.	Will the system adapt to the presence of turkeys?
2.1.4	Deceleration Time	Stop/slow to < 0.05 m/s within 2 seconds from max speed.	Stopwatch trials from top speed to full deceleration.	Can the system stop in time for unexpected events?
2.2.1	Displacement Rate	Turkey displacement > 0.5 m/s when within 5m.	Field testing with live birds (requires final concept).	Will the turkeys exit the zone fast enough?
2.2.2	Bounded Pursuit	Turkey travel < 4m toward center; pursuit < 40% zone size.	Observational studies during live field trials.	Will the turkeys leave the zone while being chased?
2.2.3	Zone Coverage	Positional resolution 0.5m; angular resolution 10°.	Test UWB anchor accuracy against physical measurements.	Will the system know where it is located?
User Functions				
3.1	Manual Input Latency	< 1s motor input response; < 0.5s Emergency Stop response.	Physical testing of the button and UI responsiveness.	Can the users give custom inputs effectively?
3.2	Device Tracking	Geolocation error < 1m; 1 minute data update cycle.	Field test final product coordinate reporting.	Can the user find the exact location of the device?
3.3	Video Display	Min resolution 240p at 5 frames per second.	Visual inspection and frame rate measurement on the UI.	Can the user clearly view the interface?

The following section provides detailed methods and results for the verification and validation of each specification.

Turkey Detection Accuracy (Req 1.1.1)
Turkey detection accuracy was verified by generating a confusion matrix based on a hybrid dataset, consisting of 10% locally recorded campus imagery and 90% curated data from the internet. This dataset composition achieved a true positive identification rate exceeding 90% for the North Campus environment. To validate performance across various lighting and background conditions, the team deployed the robot at fixed waypoints and utilized a turkey decoy at distances of 5, 10, and 15 meters. These trials were repeated over multiple dates from morning to evening to account for lighting variance. Software logs recorded the number of frames where a turkey was correctly identified against the total frames processed, confirming the true positive rate consistently exceeded the 90% threshold. Furthermore, observational testing during live turkey encounters successfully confirmed the system's ability to detect real birds.

Bounding Box Accuracy (Req 1.1.2)
To verify the Intersection over Union for humans, the first step was to create a standard to compare the performance against. Multiple videos were taken with the onboard camera of people walking at various speeds and angles within 1-5 meters, referencing distance markers for exact positioning. Next, we manually labeled every 20th frame of footage with bounding boxes in the format (x, y, width, height), which was designated as the Ground Truth. Finally, a mean IoU was calculated through running a Python script comparing the spatial overlap from the Ground Truth to model predictions. Across all trials, the mean IoU remained greater than 0.7, successfully verifying the vision systems human tracking. Figure 36 shows a frame of footage from the model tracking a human and decoy turkey



Figure 36. Bounding boxes of human and decoy turkey from model in designated zone

Zone Scan Time (Req 1.2.1)
The Zone Scan Time was verified by operating the deterrent through its programmed waypoint sequence, utilizing a stop and go rotation pattern at each node to achieve a full 360 degree field of view. To ensure the vision system had sufficient time for inference without motion blur, the device rotated in incremental steps. A stopwatch and visual confirmation was used to record the duration of each full cycle. We conducted three trials each in distinct areas of the zone to account for changes in terrain, which allowed us to calculate an average rotation time. These trials confirmed that surface variations and fixed obstacles did not impede the drivetrain's performance or the navigation stack's ability to maintain the patrol schedule. The system completed the full scan sequence consistently within the 120 second limit, validating that the robot can effectively monitor the entire zone for nearby turkeys.

Max Latency (Req 1.2.2)
Max latency was verified by benchmarking the processing pipeline on the Raspberry Pi AI hat hardware. The team developed a logging script to measure the time elapsed for a single perception cycle, which included image acquisition from the camera, preprocessing, and the YOLO model inference. Using a dataset of over 200 images of various complexities, we calculated the average and peak latency for each frame.

The verification results showed that the system maintained a classification latency of less than 1000 ms per image, even during high activity scenarios. This response validates the system can update its navigation stack based on the current position of a turkey rather than reacting to outdated environmental data. This low latency also prevented overshooting during pursuit and allowed for immediate stops when human obstacles were detected.

Computing Resources (Req 1.2.3)
The computational resource consumption was verified by running a stress test on the full software stack (CV, Navigation, and UI) on the Raspberry Pi 5. System metrics were monitored, with CPU usage averaging around 10% (below 50%) and RAM consumption averaging 20% (below 70%). This confirms that the hardware operates within stable thermal and processing limits, preventing system crashes and chronic issues in long term deployment.

Zone Confinement (Req 2.1.1)
Verification of zone confinement was accomplished through monitoring autonomous navigation trials. The device maintained established geofence boundaries of 2 meters for walkways and 4 meters from the southern roads. Pedestrians and traffic were not impacted from operation in the zone, validating the zone perimeter is safe for the community.

Obstacle Avoidance (Req 2.1.2)
Validation for obstacle avoidance was conducted through multiple human proximity trials. Team members approached the device at various speeds and angles while in autonomous roaming. Referencing physical markers, we confirmed the deterrent reduced its speed below 0.5 m/s once a human was detected within the 4 meter buffer, and maintained a minimum 1 meter clearance from all subjects. Data collection involved filming each trial, and measuring the deceleration phase with a stopwatch and measuring tape.

Max Travel Speed (Req 2.1.3)
The speeds were verified by manually triggering pursuit modes and recording the velocity with a stopwatch and distance markings. The pursuit mode was below 3 m/s, and the neutral mode was below 2 m/s, validation the system can maintain safe speeds in a public environment.

Deceleration Time (Req 2.1.4)
To verify stopping performance, the final device was manually controlled to drive at it max speed, and then prompted to immediately stop. Using stopwatch trials and video analysis, we measured the time between command signal to zero velocity. All trials achieved a full stop within 2 seconds, validating the drivetrain and motor controllers can handle emergency stop situations and minimizing risk of collisions.

Displacement Rate (Req 2.2.1)
The location of turkeys on North Campus was outside of our control, so multiple members scouted for birds on days we tested the final device. Testing the turkeys response outside the zone required manual control of the drivetrain, since our navigation system uses existing mapping and is not fully autonomous. Our best example of turkey displacement was in the parking lot near the Industrial and Operational Engineering building. When two turkeys were approached by the canine, they simultaneously gobbled and frantically ran away. The average speed of the turkeys when the canine was within 5 meters was approximately 2 m/s, which exceeds our minimum speed of 0.5 m/s. An image of the turkeys running from the canine is shown below in Figure 37.



Figure 37. Two turkeys running from the deterrent outside the IOE building at the University of Michigan's North Campus on April 11, 2026

Bounded Pursuit (Req 2.2.2)
The verification of bounded pursuit could not be completed as intended due to the absence of live turkeys in the zone during the testing window. Had birds been present, the team would have utilized video footage to verify that turkeys were displaced less than 4 meters toward the center and that pursuit sequences remained within 40% of the total zone area. Although live field data was unavailable, the underlying pursuit logic was successfully validated through simulations, confirming the system's ability to redirect targets toward the zone boarder.

Zone Coverage (Req 2.2.3)
Initial plans for device tracking involved using the UWB (Ultra-Wideband) anchoring system to track positional resolution. Unfortunately, during field testing the UWB system only functioned reliably in indoor environments, failing to provide data in the real outdoor zone.

Following the failure of the UWB verification, the team transitioned to GPS based tracking. While the GPS integration provided a reliable data update cycle, field measurements showed a location precision averaging 3 meters of error. As a result, the system failed to meet the initial requirement of geolocation error of less than 0.5 meter. Despite the failure to meet this specific metric, the system was still able to execute general patrol patterns, though the precise position within the zone was limited by the GPS accuracy.

Manual Input Latency (Req 3.1)
Verification involved measuring the time between selecting a command on the UI and visually observing the physical motor response. Both requirements were verified, with all general commands taking less than 1 second, and all emergency stop trials staying under 0.5 seconds. This further validates that the user maintains full discretionary control over the device at all times.

Device Tracking (Req 3.2)

As discussed in the Zone Coverage section, the GPS localization was not accurate enough to meet the required 1 meter specification. Because the system produced a geolocation error of approximately 3 meters during field trials, this requirement failed verification.

Video Display (Req 3.3)
The video display was verified by conducting a visual inspection of the UI while the robot was in motion. Using browser tools, we measured the incoming stream's resolution and frame reate, which exceeded 240p and 5 FPS. Additionally, we validated that multiple users had sufficient visual clarity on the stream to identify turkeys and monitor the device's surroundings through the interface.

Drivetrain Prototype
Given the nature of the project, it would be unrealistic to wait for the drivetrain and canine design teams to finish their portions of the system before testing our control logic and wiring. To work around this, our subteam decided to design and construct a prototype system, using smaller motors that can interface with our drivers, and room to mount the computer vision and microcontroller components. An image of the prototype is shown in Figure 38 below.

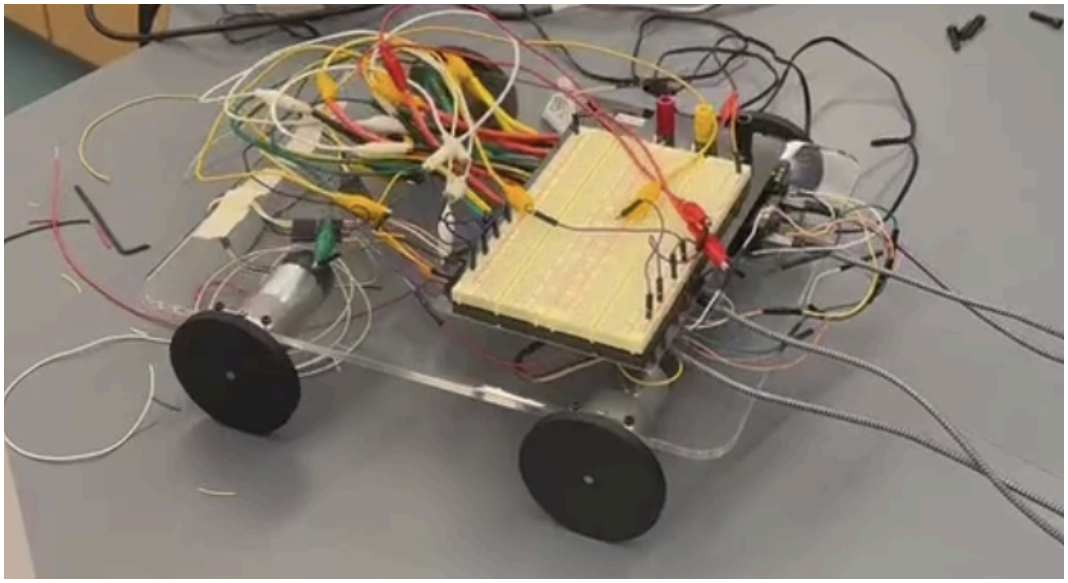


Figure 38. Image of prototype drivetrain used for control validation

The prototype consists of a laser cut acrylic baseplate, mimicking the baseplate of our alpha design, with four 37D Pololu 30:1 motors mounted that interface with our final motor drivers. The wheels we used were 3D printed using PLA, and are much thinner than the actual treads for ease of manufacturing and assembly.

17 PERFORMANCE DEMONSTRATION

After assembling and integrating our final prototype, our team conducted a series of tests to demonstrate and evaluate the effectiveness of our project. Based on outdoor field testing, our control system allowed for successful maneuvering of various terrains using our tank drive approach, both in manual and autonomous mode. Our design was also able to reliably control the canine module state, transitioning seamlessly based on our zone coverage conditions and visual inputs. Manual inputs, such as drivetrain speed and the emergency stop yielded a response time of less than 0.5 seconds, residing within our stated specifications.

The primary bottleneck for our turkey deterrence pipeline performance was object detection accuracy. While we collected promising data from our testing dataset results (over 90% true positive), the actual in-field performance was far worse than our expected accuracy, both on real turkeys and the turkey decoy (see Figure 39 below). While frame by frame data was difficult to obtain, we observed that our model frequently failed to detect turkeys in frame, and often misidentified static objects (leafs, rocks etc.) as turkeys. This can largely be attributed to poor lighting conditions (cloudy, dark etc.), which our model was not trained for, in combination with latency issues that stem from our hardware limitations.

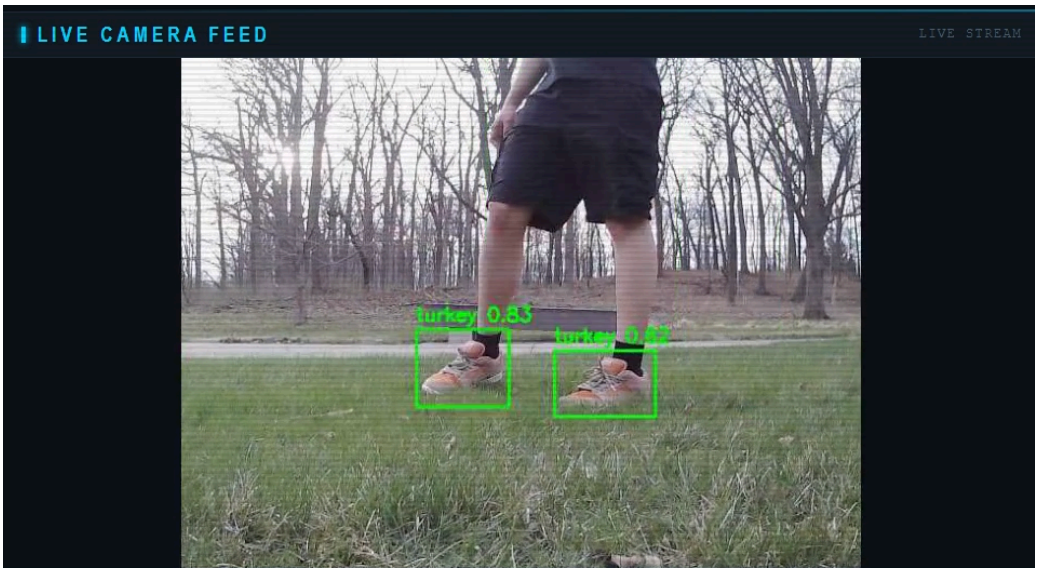


Figure 39. Image of our computer vision UI providing false positive identification of a pair of shoes as a turkey during field testing.

In addition to object detection, our computer vision distance estimation has large accuracy errors as well, of up to 3 m from dynamic objects. While the approach of using only cameras and pixel data is used in industry, such as with self-driving vehicles, we require more cameras and a far more robust data set in order to ensure that our distance estimation is sufficiently accurate.

Aside from vision, our navigation system was able to provide the functionality that we required, as long as we calibrated rigorously beforehand (this means position in the zone, as well as direction). When doing so, our robot successfully and accurately traversed between waypoints, avoided obstacles based on our prespecified mapping system, and stayed within the confinements of our zone boundary, which can be seen in Figure 40 below.

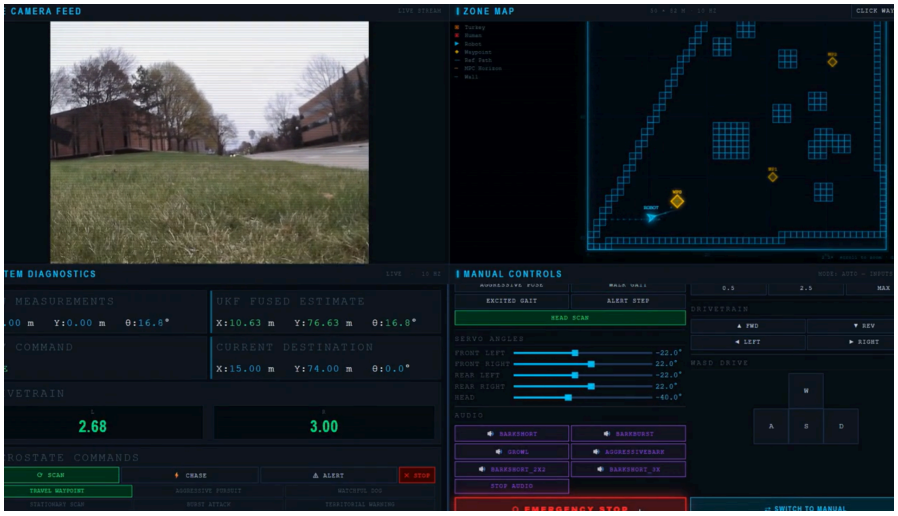


Figure 40. Image of UI with robot location mapped on to static map, and robot traversing the zone to waypoint 1.

It should be noted that if the current navigation approach is kept, a higher precision GPS module, and reduction of the grid square sizes will allow for more accurate location tracking and less conservative obstacle boundaries. During testing, our robot was able to traverse the zone waypoints within 120 seconds, but tracking accuracy frequently deviated from the desired 1m error.

While the limitations of our computer vision model made it difficult to validate our robot in autonomous mode, we were able to use manual controls on real turkeys that were spotted on campus in order to verify the effectiveness of our drivetrain maneuverability in combination with the canine module controls (see Figure 41 below).



Figure 41. Image taken during outdoor testing, chasing turkeys away in manual drive mode.

In all of the trials that we conducted, we were able to successfully pursue the turkeys and drive them away from parking lots and building entrances.

18 DISCUSSION

The following sections reflect on the design and execution of the project, identifying key limitations encountered across the software stack and offering recommendations for future iterations. Commentary spans the full scope of Team 17's responsibilities, including computer vision, navigation, controls, system integration, and project management. Where applicable, proposed changes are grounded in lessons learned from field testing and system-level integration challenges encountered over the course of the semester.

18.1 Subteam Scoping and Problem Definition

The three-subteam format presented the unique challenge of simultaneously defining both the overall project scope and the scope of each subteam. To preserve the structure required for ME450, we were shoehorned into a final design concept relatively early. The course groupings defaulted to two mechanical teams and one software team, which we largely kept for convenience. Had this project been done outside the course, we would have invested more heavily in problem definition before dividing into subgroups.

A major source of uncertainty was the turkey fear response to canines. Key questions included how critical canine movement is, the impact of pursuit speed, and how visual versus other sensory cues compare in effectiveness. While literature on the subject is sparse, we did purchase a canine decoy early on with the intent of answering some of these questions through field testing. Due to insufficient time and coordination, we were unable to do so before defining our

subteam scope and mechanical designs. From Team 19's validation field testing, they were able to scare turkeys simply by carrying their canine assembly and walking slowly. As they didn't use any servo actuation or audio deterrents, this suggests that replicating static canine appearances while closing the distance to turkeys matters more than mimicking canine behavior through servo gestures. From additional system field testing, pursuit speed proved less significant than simply closing distance, yet Team 18's aggressive speed and acceleration requirements led to drivetrain overengineering with likely minimal impact on turkey deterrence.

The final deliverable had a significant overemphasis on mechanical design at the expense of autonomy and algorithmic development, which is attributed to this being a mechanical engineering capstone course. At the time, Team 17 did not fully appreciate the technical challenges entailed by the software-focused role. Computer vision, navigation, and controls are each rich enough to warrant a dedicated team, yet all defaulted to our responsibilities, spreading us thin. Given the potential overengineering on the mechanical side, reallocating headcount toward these software challenges would have been beneficial. While outside the scope of ME450, the optimal team composition would have fewer mechanical engineers and more computer scientists and robotics engineers.

18.2 Design Critique

While significant progress was made within the semester timeline, the design carries several notable limitations. These include vision-related issues at both the model and hardware levels, limitations in navigation scope, and areas within the control system where performance could be improved. Recommendations for addressing each of these areas are outlined in the subsequent sections.

18.2.1 Computer Vision

Computer vision was an ambitious portion of our subteam scope. We had initially considered using *SpeciesNet*, which is a model trained to identify all species at high accuracy. However, estimated frame rates of <0.2 FPS while run in real-time on our personal laptops quickly ruled that option out. Hence, we opted to construct our own vision model to classify turkeys in real-time using a YOLO framework. While the end result of our model was somewhat functional in detecting turkeys and humans, it was not sufficient at the range nor accuracy defined by our requirements. This is primarily attributed to insufficient classification data, communication latencies, and camera mounting locations, which are discussed in the subsequent sections.

Data is critical for machine learning applications, particularly to account for variations in lighting, angle, and environmental noise. Publicly available turkey datasets are minimal, while human classification data is abundant. This discrepancy is reflected in our model as human detection noticeably outperforms turkey detection. Without a significant public turkey dataset, we relied primarily on photos and videos taken by the team, with hand-labeled classifications.

This process was slow and severely limited by the volume of data we were able to collect. In retrospect, we should have pursued external data collection efforts such as university outreach or a turkey photo competition, along with exploring alternatives to reduce the workload burden of manual classification.

A related issue was classification speed. Running the computer vision model on the Raspberry Pi consumed significant CPU resources, reducing frame rate to approximately 2 FPS while slowing all other processes. We had planned to alleviate this by purchasing a Raspberry Pi AI Hat, an onboard GPU predicted to achieve frame rates above 20 FPS with minimal CPU impact. Procurement was delayed due to long lead times, and after receiving the AI Hat late in the project, we encountered significant issues converting our model to a compatible format. Specifically, the pipeline required converting from .pt to .onnx to .hef to .har, a process that is poorly documented and which we were ultimately unable to complete successfully. As a workaround, we offloaded inference to a laptop by sending raw images from the Pi over WiFi and returning classification results. This reduced CPU load on the Pi but introduced bandwidth constraints, capping performance at approximately 4 FPS with our 640x480 model. The resulting low sampling rate made turkey detections harder to obtain and maintain.

A third anticipated issue is camera placement close to the ground, which reduces the viable field of view and limits the range at which turkeys can be detected. Since most of our training data was captured from a shoulder-level vantage point, the model may struggle to reliably detect turkeys from a 6-inch height. Being rigidly mounted to the drivetrain, the camera may also experience more vibration and jitter than anticipated. A higher-quality camera with a wide-angle lens or motion blur compensation could help mitigate some of these effects, though our design was constrained to USB-compatible cameras, which largely limited us to standard webcams.

It is unclear which of these issues is the primary source of failure in our vision system. Isolating and addressing them would require additional time and testing. Further investigation into our software integration pipeline may also help surface any data processing errors stemming from type mismatches or file transfer issues. Fundamentally, our model would ultimately benefit the most from classifying and training additional data, but adjustments in processing speeds or increased angle vantage point may be a sufficient last-minute solution .

18.2.2 Navigation

Navigation is likely the area requiring the most improvement before our prototype could be marketably deployed in the field. To meet our project goals within the available timeline, we reduced scope to operation within a predefined zone. Ideally, the robot would roam freely within a municipality, self-select points of interest, and map new environments in real time. We deemed this full scope unsafe to implement in a single semester and instead focused on lower-level

components of the motion planning stack, primarily localization, obstacle avoidance, and trajectory optimization.

We struggled to precisely estimate our position within the zone. Our available sensors included a GPS unit quoted at 2-3 meter accuracy, an accelerometer, and encoder readings. While we designed a preliminary UKF for sensor fusion, we did not have sufficient time to tune it adequately, which would have required field testing and measured position covariances. Instead, we relied primarily on IMU heading during turns and encoder readings for forward and backward traversal. Due to these simplifications, drift accumulation posed a safety concern, as the device could physically leave the zone into an adjacent street while still reporting a position within bounds. As a result, we required close supervision during all field operations.

Within our map, we identified static obstacles such as rocks and trees ahead of time using GPS coordinates stored in a CSV file. This was functional but tedious and not scalable for quickly transitioning to new zones. Dynamic obstacle locations were approximated using distance estimates derived from computer vision bounding boxes. Relying on a single camera, combined with the vision issues described above, resulted in unreliable obstacle avoidance. Given perfect information, our A* implementation is correctly implemented and robust in simulation. We added additional optimizations to increase buffer distances from obstacles and center the path in corridor scenarios to help compensate for localization inaccuracies.

The trajectory tracking algorithm used MPC to follow paths generated by A*, and was largely robust in producing feasible control inputs while closely tracking the planned trajectory. Encoding hard motor speed constraints explicitly also helped avoid the integrator windup and actuator saturation issues common in PID controllers. One weakness of the MPC implementation was the lack of explicit obstacle handling. While solutions such as control barrier functions and log-norm costs exist in literature, our implementations proved computationally expensive, pushing sampling time from 0.2 seconds to nearly 1 second. This latency would result in replanning once every 3 meters in worst-case scenarios, which was deemed unacceptable. In practice, the risk of catastrophic collision is predicted to be low given the conservative obstacle buffering applied during map construction. In future iterations, we would like to explore hard-constraint obstacle handling mechanisms that preserve a reasonable sampling time.

The core navigation machinery is largely suitable for future iterations. In simulation with perfect positional sensor and drivetrain actuator assumptions, we successfully replicated turkey chase sequences at up to 5x our actual operating speed, indicating algorithmic robustness. The primary limiting factor in practice is localization. We would recommend investing additional time in tuning the UKF and exploring higher-accuracy GPS or similar positioning mechanisms. The UKF algorithm is likely the better long-term approach, but validating that claim requires further tuning and field testing.

18.2.3 Controls

Controls was closest to our team's initial expertise and was the strongest of the three software branches. It focused on servicing Team 18's drivetrain through DC motor and encoder PID control, and Team 19's canine assembly through open-loop servo control and audio commands. Both modules performed reliably in field testing and sustained over an hour of exhibition at our demo without software timeouts or failures. Further commentary on specific improvements to the canine and drivetrain controls is outlined below.

A major deliverable was enabling Team 19's canine to perform servo-based gestures and audio sequences. The architecture supported asynchronous timing and real-time interrupts, and was designed to be easily configurable around Team 19's specifications. We structured the gesture system around two layers: micro gestures, defined by servo states of fixed, oscillate, and random, and macro gestures such as walk animations and poses. Separating these layers through asynchronous function calls proved effective and kept the system organized, integrating cleanly with our user interface. Audio commands were similarly designed to interrupt ongoing sounds, enabling smooth transitions between states. The modular design also made it straightforward to add new macro states and audio clips as Team 19's requests evolved. Beyond early signal instability resolved by adding a hardware servo driver, the canine module was robust throughout field testing and the final demo.

Drivetrain controls were primarily offloaded to *BasicMicro* motor drivers, which handled low-level motor and encoder communication. Our module focused on interfacing with their command protocol to read encoder speeds and send motor commands. The base implementation was largely robust, though we observed tire slipping in the field due to uneven terrain and long grass. Adding a ramp function in place of step commands for larger speed inputs improved handling. Looking ahead, we would recommend exploring traction control improvements from vehicle controls literature, as well as develop a more accurate drivetrain model to help the navigation stack estimate angle deviations from motor inputs. Both changes are within our current skill set, but are not pursued due to lack of sufficient time to implement.

18.2.4 System Integration

System integration was particularly challenging, both within our software subteam and across all three subteams at the deliverable level. To ease the communication burden, we took on the design of mechanical housing units for our electronics and camera mounts, and led most of the wiring between the drivetrain, canine actuators, microcontroller, and sensors. Our wireless user interface proved valuable in both field testing and the final expo. While integration was ultimately successful, there are numerous optimizations and process refinements that would accelerate this in future iterations.

Wiring was initially divided by domain: Team 18 handling drivetrain motors, encoders, and battery; Team 19 handling servo motors; and Team 17 handling the Raspberry Pi, GPS, and IMU. While logical in principle, an incident where inadequate wiring from Team 19 caused a burned battery prompted our subteam to claim ownership of their wiring. Team 18's encoder wiring also proved more technical than expected and similarly fell to us. Using detailed wire diagrams and component specs, we were able to clean up and properly secure the wiring with only a few minor issues. Given our role as the software team, we should have offloaded more of this work to the other subteams. We had proposed a dedicated wiring subteam drawing members from all three groups, and while a preliminary wiring diagram was produced, the effort lost organization toward the end of the project. The time spent on wiring came directly at the expense of algorithmic testing and refinements, likely contributing to some of the software shortcomings described above.

We constructed a user interface to serve as both a debugging tool and a remote actuation mechanism for field operation. It ultimately proved to be a highly effective demo tool: members of Team 18 and Team 19 were able to operate the full prototype at the expo with minimal instruction from us. The UI included diagnostic features such as a live camera feed and video map that accelerated debugging, as well as an emergency stop that saw heavy use during full-system testing. Every major component of the UI found use at some point during the project, and we would recommend a similar interface in future iterations.

18.3 Project Risks

The project posed significant risks due to its aggressive scope and tight timeline. We mitigated supply risk through early and proactive ordering to account for long lead times. A second challenge was bridging technical knowledge gaps, as our primary operating domain is robotics and software rather than mechanical engineering. A third was coordinating with the other subteam leads to ensure proper integration across the full system. The subsequent sections outline our strategies for addressing these challenges and their effect on the final deliverable.

Given the tight timeline, we adopted an aggressive ordering approach, trading money for time wherever possible. When faced with hardware decisions, our default was to buy both options, try them in parallel, and move on while waiting on lead times. Our localization sensor selection illustrates this well: we cycled through a budget GPS, UWB anchors that proved ineffective outdoors, and ultimately a higher-accuracy GPS. Getting hardware into our hands early allowed us to design software around real components rather than assumptions, significantly derisking integration. Had we waited for finalized mechanical assemblies before ordering, we would have been caught in lead times at a critical stage of the project.

The primary risk from the outset was the steep learning curve required to complete this project. Rather than exhaustively exploring the solution space during scoping, we prioritized hands-on

experimentation with relevant toolchains and practical implementation. While this meant committing to an approach before fully evaluating all alternatives, it accelerated our foundational understanding and helped us identify early which components would be straightforward versus difficult, allowing us to allocate time and resources accordingly. This proved particularly valuable in bridging knowledge gaps in localization, path planning, trajectory optimization, and computer vision, and helped us establish internal milestones to stay on pace outside of the typical ME450 course assignment deadlines.

Integration was an anticipated risk from early on. With three teams contributing moving mechanical parts, extensive software, and drivers bridging the gaps, we expected significant difficulties in the week leading up to the expo. To prepare, we were deliberate about testing software in isolation as early as possible. For example, we built simulations of motor dynamics to test navigation algorithms before full integration, and validated canine servo and audio commands independently through the UI. This modular testing approach allowed us to identify and resolve many software bugs before full system assembly, enabling us to get the device operational relatively quickly and avoid a team-wide deadlock of simultaneous hardware and software debugging.

19 RECOMMENDATIONS

If this project were to undergo a new iteration, several architectural recommendations are proposed to improve its viability and scalability. These largely concern expanding the project scope and identifying the tools and expertise necessary to support that expansion. Each recommendation is outlined in the subsequent sections.

19.1 Expanded Autonomy Functions

The sponsor's end vision is a device that autonomously patrols municipalities, navigating to and deterring turkeys from designated hotspots. Achieving this at scale would require significant further development in the navigation stack, including self-arbitrating waypoints, road collision avoidance, and improved turkey targeting and dynamic chase behavior. Our design establishes the underlying machinery through A* path generation and MPC trajectory optimization, but the higher-level algorithms required for autonomous patrol demand domain expertise we were unable to acquire within the project timeline. Safety is also a significant concern, requiring a substantial overhaul to prevent dangerous behaviors such as obstacle collisions, navigation into unsafe zones, or misidentifying people as turkeys. We recommend that future iterations bring in engineers with dedicated robotics expertise to address these gaps, as we recognize the limitations of our background in charting a precise path forward for the level of autonomy our sponsor envisions.

19.2 Vision Hardware and Online Obstacle Detection

Our architecture relied on a single camera, which led to vision inaccuracies, and pre-mapped static obstacles, which are both tedious to map and inaccurate. To address these limitations, we recommend expanding the vision hardware in two ways. First, adding cameras to cover blind spots and provide multiple viewing angles would improve classification accuracy and enable safer maneuvering with full situational awareness. Second, pre-mapping obstacles is infeasible at municipal scale, so we recommend incorporating a 3D LiDAR camera paired with a simultaneous localization and mapping (SLAM) algorithm to map the environment in real time. We opted against this approach given the technical complexity and integration risks, but a viable SLAM implementation would significantly improve obstacle avoidance and usability in future iterations. Expanding camera coverage would also demand greater computational power, necessitating either a functional Raspberry Pi AI Hat or a more capable processor altogether.

19.3 GPS Tracking

Our GPS sensor was a relatively affordable Amazon purchase that, while functional, had notable limitations. Lock-on time after exiting a building was slow, and the sensor required open skies to operate reliably. In a municipal deployment where the robot may frequently pass under trees or structures, a higher-quality GPS would provide more robust signal acquisition. We opted for an open-source GPS for ease of code integration, though it was harder to tune as a result. If the project transitions to LiDAR-based obstacle avoidance, precise GPS localization becomes less critical, with GPS serving primarily for long-term user monitoring. In that scenario, a closed-source GPS with a self-contained wireless interface would be the superior option.

19.4 ROS2 Framework

ROS2 is a widely used middleware framework for robotics that integrates sensors and actuators with various algorithms from SLAM, path planning, and control. While we were aware of its capabilities, the learning curve would have consumed roughly a month, which was unacceptable given our timeline. Instead, we implemented a bare-bones architecture that prioritized modularity and familiarity, as we felt more comfortable building and modifying our own system than adapting to the ROS2 framework. In a future iteration, particularly one led by engineers already proficient in ROS2, migrating to it would provide a wealth of robust out-of-the-box algorithms and communication tooling.

19.5 Power Stations for Automatic Charging

Battery life is a major limitation of our design. In field testing, we observed roughly 1 hour of continuous operation for the servo system, 4 hours for the drivetrain, and 3 hours for the Raspberry Pi. While sufficient for demonstration purposes, this falls short of the extended deployment periods needed for municipal use. We recommend developing a power station solution enabling automatic recharging, whether through wireless charging or autonomous battery replacement, both of which would require dedicated research and development.

Regardless of implementation, improving power longevity is a necessary step toward a viable final product.

19.6 Wire Professionalism and Ease of User Operation

Given the prototype nature of our design, numerous decisions prioritized functionality over form factor. Wiring was managed with off-the-shelf mounts and 3D prints, and circuitry relied on flimsy wire connections that occasionally came undone. For a more polished iteration, we recommend integrating wire harnessing directly into the mechanical design through grooves, slots, and grommets, and transitioning to custom PCBs for the electronics. We also recommend adding dedicated covers for the drivetrain and servo batteries, as exposed high voltages pose a meaningful safety risk.

Our software was built for functionality rather than ease of use by non-technical operators. A significant pain point was remote access to the Raspberry Pi in environments with limited WiFi. Key improvements would include an autoboot script to launch the system without manual intervention, a more robust wireless architecture by integrating a dedicated WiFi module into a PCB rather than relying on a separate ESP32, and streamlined UI connectivity. Our software is also prone to faulting under poor wiring conditions and hardware instabilities, so adding safety routines, diagnostics, and logging would improve reliability and development speed. More broadly, a user manual and further abstraction of core software functions would make the system more accessible to non-technical operators.

20 REFLECTION

What began as a local challenge to deter wild turkeys evolved into a broader exploration of public safety, autonomous ethics, and equitable teamwork. This reflection details the global, societal, interpersonal, and ethical dimensions that shaped our final design.

20.1 Global & Societal Impact

At the beginning of this project, our primary focus was on the immediate technical challenge of deterring wild turkeys on the University of Michigan North Campus. However, as the project progressed, our perspective broadened to recognize the wider societal footprint of deploying an autonomous, vision-enabled robot in public spaces.

20.1.1 Public Health, Safety, and Welfare

The core purpose of the robot is to protect public welfare by mitigating aggressive wildlife encounters, preventing property damage, and keeping walkways clear. However, safety was our top priority; our Team 17 intelligence system had to guarantee that the robot would accurately differentiate between turkeys and humans to ensure it never inadvertently harassed a pedestrian.

20.1.2 Global Context

While designed for a localized issue in Michigan, the concept has significant global market potential. Urban wildlife encroachment is a worldwide challenge: from Canada geese in North

American parks to urban macaques in Southeast Asia. An autonomous, humane hazing robot provides a scalable, non-lethal alternative to pest control globally.

20.1.3 Social Impacts

For manufacturing, sourcing electronic components (cameras, microcontrollers) involves complex global supply chains that often have hidden social costs regarding labor. During usage, the design shifts the burden of wildlife management from manual human labor (facilities staff) to an automated system, potentially freeing up labor for more meaningful tasks while altering how the campus community interacts with public spaces. In disposing, E-waste is a critical social issue. The disposal of lithium-ion batteries and PCBs requires responsible recycling infrastructure to prevent toxic runoff in local communities.

20.1.4 Economic Impacts

Before manufacturing, the initial capital cost of sensors, compute modules, and drivetrain components is relatively high for a prototype. On the other hand, during usage, long-term economic benefits include reduced maintenance costs for U-M facilities, fewer property damage claims, and reduced staff hours spent chasing birds. In disposing of the robot, decommissioning the robot incurs costs associated with specialized e-waste recycling.

20.1.5 Basic Tools Used

We utilized stakeholder maps to identify and prioritize the needs of the DNR, U-M maintenance staff, pedestrians, and the turkeys themselves. We also considered basic life cycle costing to weigh the upfront cost of the intelligence hardware against the long-term operational savings of an autonomous system.

20.2 Inclusion, Equity, and Team Dynamics

Working in a highly integrated project with three distinct subteams required intentional navigation of interpersonal and power dynamics to ensure equitable collaboration.

20.2.1 Team Dynamics and Identity

Within Team 17, and in our interactions with Teams 18 and 19, our diverse engineering backgrounds (e.g., software focus vs. hardware focus) and stylistic differences heavily influenced our approach. Students with stronger coding backgrounds naturally held more technical privilege in the designing process. To balance this power dynamic, we adopted a modular workflow where team members could lead sub-tasks based on their strengths, while actively cross-training others.

20.2.2 Stakeholders and End Users

As students, our daily experience with the turkeys is largely as pedestrians navigating around them. However, our primary end-users and stakeholders are the U-M maintenance staff and the DNR, who bear the operational and environmental burden of managing the flocks. This difference in lived experience required us to actively de-center our own mild inconveniences and design for the practical, labor-intensive realities of facilities management. We prioritized a

user-friendly wireless interface specifically because the end-users require intuitive, low-friction control over the robot without needing a computer science background.

20.2.3 Power Differences with the Sponsor

Our sponsor, Jared Duquette of the DNR, holds significant domain expertise in wildlife biology and the ultimate authority over the project's direction. There was a clear power differential: he is the professional stakeholder, and we are the student engineers. We navigated this by establishing a symbiotic relationship. We deferred to his expertise regarding what constitutes "humane hazing" and the behavioral metrics of turkeys. In turn, we exercised our engineering agency by communicating the technical boundaries of computer vision and autonomous navigation, ensuring that his biological requirements were translated into feasible technical constraints.

20.3 Ethics

The deployment of an autonomous, mobile robot equipped with cameras in a public space inherently introduces profound ethical considerations.

20.3.1 Ethical Dilemmas in Design

Our primary ethical dilemma was balancing effective deterrence with the humane treatment of the animals. We had to ensure the robot was intimidating enough to move the turkeys, but not so fast or aggressive that it caused them physical harm or undue stress. Furthermore, from an intelligence perspective (Team 17), we faced the dilemma of algorithmic accuracy. A false positive, where the robot mistakes a small dog for a turkey, could result in the robot hazing a civilian's pet. We managed these dilemmas by implementing strict fail-safes, prioritizing conservative bounding-box distance estimations, and hardcoding speed limits and boundary zones to restrict the robot's operational radius.

20.3.2 Potential Marketplace Issues

If this product were commercialized, the most glaring ethical issue would be privacy. A robot continuously recording video to run computer vision algorithms in public spaces poses a surveillance risk to pedestrians. Additionally, there is a risk of misuse; users might modify the high-level controls to intentionally harass domestic animals or people. Implementing on-edge processing (where footage is processed locally and immediately discarded rather than stored in the cloud) would be a necessary ethical safeguard for market release.

20.3.3 Personal vs. Professional Ethics

Our personal ethics aligned closely with the professional ethics expected by the University of Michigan and future engineering employers. Professional codes of ethics prioritize the health, safety, and welfare of the public. While our personal ethics provided the internal motivation to care about these issues, the professional ethics framework provided the rigorous, formalized

structure required to actually implement them, ensuring that we validated our computer vision models rigorously and documented our safety protocols comprehensively.

21 CONCLUSION

The University of Michigan North Campus community has been experiencing conflict with wild turkeys that harass pedestrians, obstruct building entrances, and damage property. Our mission is to humanely deter the turkey population with an autonomous robot through hazing tactics. Team 17 is specifically responsible for the system's intelligence, which includes computer vision, navigation, and high level control. Teams 18 & 19 are leading the drivetrain and decoy design.

By the end of our semester, our team was able to successfully design and construct a full vision-to-control deterrence pipeline for our robot dog project, fully integrated with a prototype user interface, and fully functional servo and motor controls. From a controls perspective, our final product was very successful in achieving the maneuverability that our drivetrain was designed for, and transitioned seamlessly and automatically between canine states that Team 19 designed, in accordance with our specified conditions. Although dependent on rigorous calibration, our navigation system worked well for autonomous patrolling of the designated zone, and was able to avoid static obstacles that were predefined in our map. Assuming perfect visual information and object identification, we believe that our robot would be able to effectively deter turkeys from our zone, with minimal human interference.

Through our testing and validation stages, our team identified that the main bottleneck of our full turkey deterrence pipeline was the computer vision detection accuracy. We used a testing dataset to quantify our projected performance, however, field testing results deviated significantly from our expected accuracy. By extension, our distance estimation algorithm's accuracy also suffered, which was crucial for path planning with dynamic objects. If we had 2 more weeks to work on the project, our focus would be on improving our CV performance by artificially expanding our training data in different lighting conditions, and also upgrading the hardware to decrease processing latency. With these improvements, we expect the autonomous mode to operate more smoothly, despite the more difficult problem of distance estimation accuracy.

If our project is to be continued in subsequent semesters or by the DNR, our suggested next steps are to focus on improving the autonomous functionality through vision and navigation. While our distance estimation strategy using pixels is not a novel approach, we would require additional cameras and a much larger dataset to achieve the in-field accuracy that is required. Alternatively, LiDAR would be a more robust option for obstacle avoidance. Once the desired functionality is achieved, we would recommend implementing a charging/docking system, and improve the ease of user operation through wire coverings and streamlining the UI.

ACKNOWLEDGEMENTS

The team would like to express its sincere gratitude to Dr. Jared Duquette for his sponsorship and support of this project. We are also deeply appreciative of Professor Shorya Awtar for his invaluable guidance and mentorship. Special thanks are extended to the University of Michigan Maintenance department for their logistical assistance, the Mechanical Engineering Purchasing office, and teams 18 and 19 for their partnership and collaboration throughout the design and testing phases.

REFERENCES

[1] Team 29, 2025, "AI Turkey Mitigation System for UM North Campus," ME 450: Design and Manufacturing III, University of Michigan, Ann Arbor, MI.

[2] Berger-Tal, O., Blumstein, D. T., and Swaisgood, R. R., 2020, "Conservation Translocations: A Review of Common Difficulties and Promising Directions," *Anim. Conserv.*, **23**, pp. 121–131.

[3] Gulotta, N. A., et al., 2025, "Testing the Human Shield Hypothesis: Female Wild Turkeys Have Greater Fitness Near Human Activity," *J. Appl. Ecol.* (In Press).

[4] Morelli, F., et al., 2022, "Flight Initiation Distance and Refuge in Urban Birds," *Sci. Total Environ*., **838**, p. 156554.

[5] Coates, R. W., et al., 2010, "Evaluation of Damage by Vertebrate Pests in California Vineyards and Control of Wild Turkeys by Bioacoustics," *Hum.-Wildl. Interact.*, **4**(1), pp. 130–145.

[6] Miller, J. E., 2018, "Wild Turkeys: Wildlife Damage Management Technical Series," USDA, APHIS, WS National Wildlife Research Center.

[7] Bird Control Group, 2024, "Automated Laser Bird Repellent," Bird Control Group. Available: https://birdcontrolgroup.com/automated-laser-bird-repellent/

[8] Bird-B-Gone, 2026, "Motion Activated Bird Scarecrow Sprinkler [Photograph]," Bird-B-Gone. Available: https://www.birdbgone.com/motion-activated-bird-scarecrow-sprinkler/

[9] Frawley, B. J., 2024, "2024 Michigan Spring Turkey Hunter Survey," Michigan Department of Natural Resources.

[10] Pahl, G., and Beitz, W., 2007, Engineering Design: A Systematic Approach, Springer-Verlag, Berlin.

[11] Center for Socially Engaged Design (CSED), 2024, "Socially Engaged Design Academy (SEDA) Learning Blocks," University of Michigan.

[12] "Design Process Overview," 2026, ME 450 Learning Block Module, University of Michigan.

[13] ANSI/RIA, 2020, "Industrial Mobile Robots Safety Requirements Part 1: Requirements for the Industrial Mobile Robot," ANSI/RIA R15.08-1-2020.

[14] ISO, 2014, "Robots and Robotic Devices Safety Requirements for Personal Care Robots," ISO 13482:2014.

[15] IEEE, 2022, "IEEE Standard for General Requirements of Online Detection Based on Machine Vision in Intelligent Manufacturing," IEEE 2671-2022.

[16] "Design Context Overview," 2026, ME 450 Learning Block Module, University of Michigan.

[17] University of Michigan, 2024, "Standard Practice Guide 303.04: Intellectual Property and Commercialization."

[18] Michigan Department of Natural Resources, 2024, "Wildlife Damage and Nuisance Control Permits: Resident Canada Goose and Nuisance Wildlife Management Programs."

[19] National Wild Turkey Federation, 2024, "The NWTF Mission: A Cornerstone of Conservation."

[20] Shah, D., 2023, "Intersection Over Union (IoU): Definition, Calculation, Code," V7 Labs [Online]. Available: https://www.v7labs.com/blog/intersection-over-union-guide.

[21] Li, Z., and Wang, Y., 2014, "Coordinated Control of Slip Ratio for Wheeled Mobile Robots Climbing Loose Sloped Terrain," *The Scientific World Journal*. Available: https://pmc.ncbi.nlm.nih.gov/articles/PMC4167808/

[21] The Kalman Filter, 2026, "Kalman Filter Explained Simply," The Kalman Filter. Available: https://thekalmanfilter.com/kalman-filter-explained-simply/

[22] Wan, E. A., and van der Merwe, R., 2000, "The Unscented Kalman Filter for Nonlinear Estimation," *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium*. Available: https://groups.seas.harvard.edu/courses/cs281/papers/unscented.pdf

TEAM BIOS



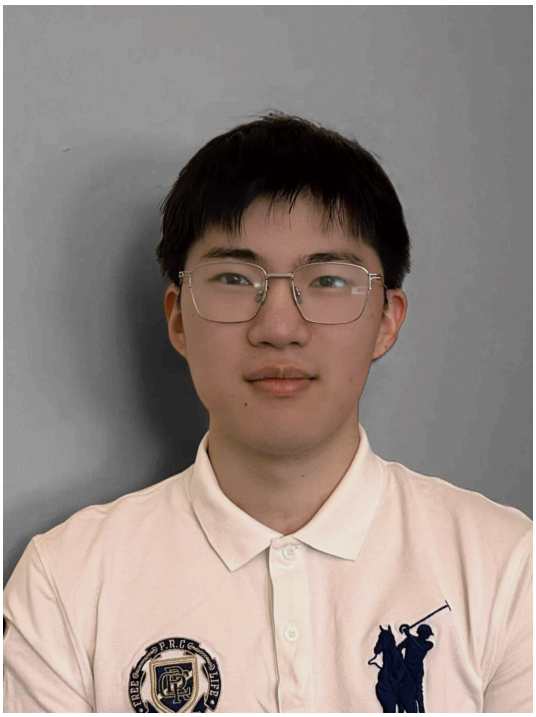
Brady Buten

I am from Troy, Michigan, and I am currently a senior studying Mechanical Engineering and Computer Engineering. I am interested in the intersection of mechanical hardware and software-based control, where physical systems are integrated with sensing, computation, and intelligent decision making. This interest developed through hands-on coursework and project experiences that combined mechanical design with programming, controls, and embedded systems. After completing my undergraduate, I plan to pursue a master's degree in Electrical and Computer Engineering at the University of Michigan, with a focus on controls and robotics. I aim to apply this multidisciplinary background to the development of electromechanical systems in the robotics or manufacturing industry. Outside of school, I frequently enjoy lifting weights, experimenting with different foods, and hanging out with friends and family.



Eric Giles

I'm from Kentwood, Michigan, and am currently a senior studying Mechanical Engineering. My interests include design, manufacturing, and automation. Future plans are to work in industry after graduation and do a part time masters program. Outside of school I enjoy working on vintage motorcycles, playing sports, and investing.



Yuxi (Ethan) Chen

I am from Shanghai, China, and I'm currently a senior student majoring in Mechanical Engineering with a Computer Science Minor. My academic interests fall on robotic system development, hardware–software integration, and learning-based control. These interests have driven me to develop strong skills in designing customized control and sensing strategies for diverse actuation systems, as well as validating these designs through hands-on experimentation. After graduation, I plan to pursue a research-oriented master's program with a focus on robotics and control. My long-term goal is to apply advanced control methodologies to emerging actuation platforms, such as soft robotic systems and microrobots. Outside of academics, I enjoy game designing, jogging, and reading novels.



Zachary Sun

I am from Vancouver, Canada, and am currently a senior in Mechanical Engineering at the University of Michigan. My academic focus up to this point has been centered around design and manufacturing, however, I wanted to use this course as an opportunity to branch out into firmware and artificial intelligence. My dream job would be a Formula 1 race engineer, or more broadly, a career in the automotive industry. After I graduate, I plan to either continue my engineering studies through a master's program, or pivot to pursuing an MBA. Outside of academics, I enjoy playing racquet sports, basketball, reading, and playing the piano.

APPENDIX

A.1 SOFTWARE FILES AND DOCUMENTATION

See attached link for a google drive containing additional documentation. Additionally, our software files are in this drive under the “Software” folder.
https://drive.google.com/drive/folders/1cpAG_d8g6CWU-wwvXUBekBTowT9CSH9v?usp=sharing

A2 BOM SECTION

Item	Quantity	Source	Cost	Contact	Notes
Raspberry Pi 5	1	Amazon	$149.99	amazon.com	
Raspberry Pi AI Hat	1	Amazon	$99.79	amazon.com	
RPi 5 Case	1	Amazon	$19.90	amazon.com	
USB Camera	1	Amazon	$18.99	amazon.com	
MicroSD 64 GB Card	1	Amazon	$20.99	amazon.com	
Micro SD Card Reader	1	Amazon	$9.99	amazon.com	
GPS Module	1	Amazon	$19.97	amazon.com	
9 DOF IMU	2	Adafruit	$29.95	adafruit.com	
Portable Charger	1	Amazon	$9.99	amazon.com	
Bluetooth Speaker	1	Amazon	$29.99	amazon.com	
DC Motor Drivers	4	Pololu	$119.95	pololu.com	
Encoder Cables	3	Amazon	$7.99	amazon.com	
Encoder adapter	4	Amazon	$7.45	amazon.com	
Buck Converter	1	Amazon	$6.99	amazon.com	5V to 3.3V
ESP32 Dev Boards	1	Amazon	$16.79	amazon.com	
Total Cost			**$996.67**		

A3 CANINE ACTUATOR SPECIFICATIONS

This document details the canine servo motor and audio actuator commands decided by Team 19 to be implemented by Team 17 in software. For this document, outline all modifications made by Team 19 in red.

A3.1 Actuator Design Scope

1. Any servo positions at any time, so long as they are mechanically feasible
2. Any predefined audio clips played at any time
3. Servo positions and audio can be played at independent intervals or sequentially
    a. Independent: Play barking every 3 seconds, swivel head every 4 seconds
    b. Sequentially: Play barking for 1 second. After, play growling for 2 seconds
4. Allowed to alter drivetrain speed at any time, so long as feasible in motors, and follows requirements set by Team 18
5. Unable to make changes to the drivetrain trajectory

A3.2 Audio Commands

For all audio files, update the table below. The audio format must be in a standard audio format (MP3, WAV, etc.). Feel free to delete the example entry.

Number	Name	Description	File Link
1	BarkShort	Single short bark used as a warning or attention cue	BarkShort
2	BarkBurst	Rapid bark sequence (2–3 barks) used during active chase	BarkBurst
3	Growl	Low growl indicating territorial warning	Growl
4	AggressiveBark	Continuous barking for 2–3 seconds is used during chase mode	AggressiveBark

Maximum Servo Angles and Speeds Commands

Below is a table consisting of the maximum allowed angles for each servo motor. Additionally, an optional field is specified for a maximum perturbation angular speed. If left blank, this will default to the maximum needed from the provided motor command suite.

MotorName	**Max Positive Angle (degrees)**	**Max Negative Angle (degrees)**	**Maximum Oscillation Speed From Positive to Negative Max Angle (degrees/s)**

FrontLeft	+45	-45	120 deg/sec
FrontRight	+45	-45	120 deg/sec
RearLeft	+45	-45	120 deg/sec
RearRight	+45	-45	120 deg/sec
Head	+60	-60	180 deg/sec

A3.3 Fixed Motor Commands

IdlePose

Neutral resting pose

FrontLeft: Set to 0°
FrontRight: Set to 0°
RearLeft: Set to 0°
RearRight: Set to 0°
Head: Set to 0°

AlertPose

Dog appears attentive when turkey detected outside zone

FrontLeft: Set to +10°
FrontRight: Set to +10°
RearLeft: Set to +15°
RearRight: Set to +15°
Head: Set to 0°

AggressivePose

Crouching posture when preparing to chase

FrontLeft: Set to 30°
FrontRight: Set to 30°
RearLeft: Set to 0°
RearRight: Set to 0°
Head: Set to 0°

A3.4 Oscillating Motor Commands

WalkGait

Standard walking gait used during patrol or scanning.

FrontLeft

Initialize to +15°

Oscillate between +25° and −25° every 2 seconds continuously

FrontRight

Initialize to −15°

Oscillate between −25° and +25° every 2 seconds continuously

RearLeft

Initialize to −15°

Oscillate between −25° and +25° every 2 seconds continuously

RearRight

Initialize to +15°

Oscillate between +25° and −25° every 2 seconds continuously

Head

Random perturbation between −20° and +20° every 3 seconds

ExcitedGait

Faster leg oscillations are used when chasing a turkey.

FrontLeft

Initialize to 0°

Oscillate between +40° and −40° every 1 second

FrontRight

Initialize to 0°

Oscillate between −40° and +40° every 1 second

RearLeft

Initialize to 0°

Oscillate between −40° and +40° every 1 second

RearRight

Initialize to 0°

Oscillate between +40° and −40° every 1 second

Head

Random movement between −45° and +45° every 1 second

AlertStep

Slow stepping motion is used when the turkey is nearby but outside the zone.

FrontLeft

Initialize to 0°

Oscillate between +20° and −10° every 3 seconds

FrontRight

Initialize to 0°

Oscillate between −10° and +20° every 3 seconds

RearLeft

Initialize to 0°

Oscillate between −10° and +20° every 3 seconds

RearRight

Initialize to 0°

Oscillate between +20° and −10° every 3 seconds

Head

Oscillate between −40° and +40° every 5 seconds

HeadScan

FrontLeft: No change

FrontRight: No change

RearLeft: No change

RearRight: No change

Head

Sweep between −60° and +60° every 4 seconds

A3.5 Chase Protocol

The chase protocol occurs when a turkey is present and inside the zone boundaries.

List Audio + Servo Motor Commands to be played and when. Random selection is suitable, but would require the identification of multiple protocols to choose from

Protocol 1 – Aggressive Pursuit

Play AggressiveBark

Activate ExcitedGait

Wait 3 seconds

Head oscillates randomly between −40° and +40°

Repeat

Protocol 2 – Burst Attack

Play BarkBurst

Activate ExcitedGait

Wait 2 seconds

Head snap to +45°

Play BarkBurst

Continue gait

Continue ExcitedMotion

Protocol 3 – Growl and Chase

Play Growl

Activate WalkGait

Head move to +30°

Wait 2 seconds

Play AggressiveBark
Transition to ExcitedGait

A3.6 Alert Protocol

The alert protocol occurs when a turkey is present and outside the zone boundaries.
List Audio + Servo Motor Commands to be played and when. Random selection is suitable, but would require identification of multiple protocols to choose from

Protocol 1 – Watchful Dog
Activate AlertPose
Head: sweep -45 to +45 degrees
Play BarkShort every 5 seconds

Protocol 2 – Territorial Warning
Activate AlertStep
Play Growl
Head oscillates slowly between −30° and +30°

Protocol 3 – Short Warning Burst
Play BarkBurst
Head snap to +40°
Activate AlertStep
Wait 3 seconds
Return to AlertPose

A3.7 Scan Protocol

The scan protocol occurs when the robot is searching for the turkey. It is broken into two primary modes:

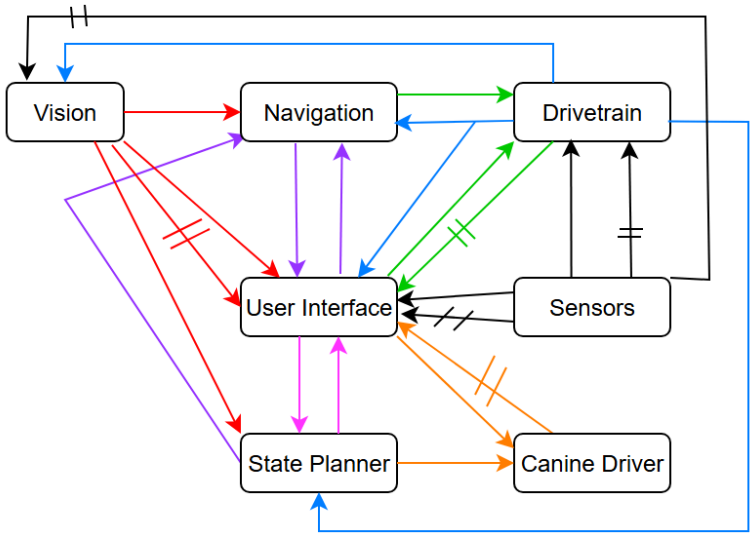
1: Traveling to a waypoint within the zone. Sequence A->B->C->... -> A
Mode 1 - Traveling between waypoints
While traveling between waypoints:
Activate WalkGait
Head oscillates between −45° and +45° every 3 seconds
Play BarkShort every 10 seconds

2: Once at a waypoint, rotate the system drivetrain X (TBD) degrees and pause. After Y rotations, travel to the next waypoint in zone
Mode 2 - Stationary Scan
Once at waypoint:

Rotate drivetrain X degrees (TBD)
Activate HeadScan
Activate AlertPose
Play BarkShort
Pause 4 seconds
Repeat until next waypoint movement

Mode 3 - Stationary Crouch
Once at waypoint:
Rotate drivetrain X degrees (TBD)
Activate AggressivePose

A.4 SOFTWARE PROTOCOL



<Vision> Red: VisionObject Protocol
<Vision> Red Striped: VideoFeed Protocol
<Canine> Orange: Canine Protocol
<Canine> Orange Striped: ServoFeedback Protocol
<Drivetrain Commands> Green: DriveCommand Protocol
<Drivetrain Commands> Green Striped: SlipFeedback Protocol
<Robot Pose> Blue: RobotPose Protocol
<State Planner> Pink: MacroState Protocol
<Navigation> Purple: MotionCommand Protocol
<Sensors> Black: UWB Protocol
<Sensors> Black Striped: Accelerometer Protocol
<Sensors> Black: UWB Protocol
<Sensors> Black Striped: Accelerometer Protocol

1A: VisionObject Protocol
Data of what is being actively observed by vision system with object classifications

Description | Turkey estimated locations on screen
Id: DynamicObjects
Turkeys: <T x 3 numpy array [Turkey X, Turkey Y, Relative angle from center of camera(radians)]>
Humans: <P x 3 numpy array [Person X, Person Y, Relative angle from center of camera(radians)]>
Ghosts: <G x 3 numpy array [Ghost X, Ghost Y, Relative angle from center of camera(radians)]

1B: VideoFeed Protocol
Description | Send Live Camera Feed to User Interface. Contains video screenshots with annotation of object detection

2A: Canine Protocol
Commands sent to the canine module to drive servos and trigger audio sounds.

2.1: Canine Pose Control
Description | Sends Pose Commands to Canine. Includes preset servo + audio sequences
Id: [“UI”, “System”]
Type: “Pose”
Data: [“IdlePose”, “AlertPose”, “AggressivePose”, “WalkGait”, “ExcitedGait”, “AlertStep”, “HeadScan”]

2.2: Raw Servo Control
Description | Direct Servo Control Commands. Canine Driver will set hard limits
Id: [“UI”, “System”]
Type: [“Servo”]
Name: [“FrontLeft”, “FrontRight”, “RearLeft”, “RearRight”, “Head”]
Data: <Floating point value of angle>

2.3: Raw Audio Control
Description | Direct commands to trigger audio clips
Id: [“UI”, “System”]
Type: “Audio”
Data: [“BarkShort”, “BarkBurst”, “Growl”, “AggressiveBark”]

2.4: UI Relinquish Command
Description | Relinguishes control back to software
Id: UnlockUI

2B: ServoFeedback Protocol
Servo motor angles published from the canine module

Name: [“FrontLeft”, “FrontRight”, “RearLeft”, “RearRight”, “Head”]
Angle: <Floating point value of angle>

3A: DriveCommand Protocol
Commands to Actuate Low Motor Drivers

3.1: UI Drive Command
Description | UI Interface to send motor commands
Id: UI
Cmd: 1 x 2 numpy array [Left Motor Speed, Right Motor Speed]

3.2: UI Relinquish Command
Description | Relinguishes control back to software
Id: UnlockUI

3.3: General Drive Command
Description | Navigation interface to send motor commands
Id: System
Cmd: 1 x 2 numpy array [Left Motor Speed, Right Motor Speed]

3B: SlipFeedback Protocol

<Float: 2> | Slip or current fault -> Red LED on UI

<Float: 1> | Slip warning -> Yellow LED on UI

<Float: 0> | All systems good -> Green LED on UI

4: RobotPose Protocol
Estimated current X, Y, and Theta of robot in space

Description | State Information Published By Drivetrain Module. Numpy array, no json encoding
<Array Data: 1 x 3 numpy array [X coord (m) ,Y coord (m), theta (rad)]>

5: MacroState Protocol
User Interface to State Planner Channel. Enable high level testing of three modes of operation. Receival of these commands puts state planner in manual mode and must be relinquished to return to auto.

5.1: Scan Sequence
Description | Puts State Decider in Manual Mode. Activates scanning routine at next waypoint
Id: Scan

5.2: Alert Sequence
Description | Puts State Decider in Manual Mode. Activates alert routine at next waypoint. No turkey spotted, will time out to after usual inactive duration
Id: Alert

5.3: Chase Sequence
Description | Puts State Decider in Manual Mode. Activates chase sequence at desired X,Y coordinates
Id: Chase

5.4: TravelWaypoint SubSequence
Description | WalkGait, Head Oscillate, BarkShort
Id: TravelWaypoint

5.5: StationaryScan SubSequence
Description | Rotate Drivetrain, HeadScan, AlertPose, ParkShort, Pause 4 || Loop
Id: StationaryScan

5.6: StationaryCrouch SubSequence
Description | Rotate Drivetrain: Activate Drivetrain, AggressivePose
Id: StationaryCrouch

5.7: AggressivePursuit SubSequence
Description | AggressiveBark, ExcitedGait, Wait 3, Head Oscillate Random || Loop?
Id: AggressivePursuit

5.8: BurstAttack SubSequence

Description | BarkBurst, Excited Gait, Wait 2, Head Snap 45, BarkBurst || Loop
Id: BurstAttack

5.9: GrowlAndChase SubSequence
Description | Growl, WalkGait, Head Move +30, Wait 2, AggressiveBark, Excited Gait
Id: GrowlAndChase

5.10: WatchfulDog SubSequence
Description | AlertPose, Heat Sweep -45 to 45, BarkShort every 5 seconds
Id: WatchfulDog

5.11: TerritorialWarning SubSequence
Description | AlertStep, Growl, Oscillate -30 to 30 slow
Id: TerritorialWarning

5.12: ShortWarningBurst SubSequence
Description | BarkBurst, Heat Snap +40, AlertStep, Wait 3, AlertPose
Id: ShortWarningBurst

5.13: Relinquish Control
Description | Puts State Decider in Automatic Mode
Id: UnlockUI

6: MotionCommand Protocol
State Planner to TrajectoryTracker Channel. Middle level to convert high level X,Y, theta goals to feasible command inputs

6.1: Update Orientation
Description | Make small adjustments to robot pose. Used for scan rotation and alert modes
Id: Orient
Cmd: 1 x 3 numpy array [X coord (m) ,Y coord (m), theta (rad)]

6.2: Traverse Zone
Description | Make significant changes to robot location. Used for chase sequence and moving to next scan waypoint
Id: Move
Cmd: 1 x 3 numpy array [X coord (m) ,Y coord (m), <unused>]

6.3: Limit Traversal Speed
Id: SpeedCap

Cmd: 1 x 1 numpy array [Motor speed (m/s)]

7A: UWB Protocol
<Array Data: 1 x 2 numpy array [UWB X coord (m) , UWB Y coord (m)]>

7B: Accelerometer Protocol
<Array Data: 1 x 1 numpy array [theta (rad)]>