EXPERIMENTAL IMPROVEMENTS TO EVENT-TRIGGERED INDIRECT HERDING CONTROL OF A COOPERATIVE AGENT

By

JAMES W. CROSS

AN HONORS THESIS PRESENTED TO THE UNIVERSITY
OF FLORIDA IN PARTIAL FULFILLMENT OF THE
REQUIREMENTS FOR THE DEGREE OF
BACHELOR OF SCIENCE MAGNA CUM LAUDE

UNIVERSITY OF FLORIDA

2026

To my Mom and Dad

ACKNOWLEDGEMENTS

These experiments would have not been possible without the extensive help troubleshooting experimental hardware by Brandon Fallin. Additionally, I would like to thank Dr. Dixon and the rest of the members in the Nonlinear Controls and Robotics group for their mentorship and support over the past four years. Finally, I would like to thank Dr. Amy for his guidance during my first three years.

TABLE OF CONTENTS

page

Abstract of Honors Thesis Presented to the Graduate School of the
University of Florida in Partial Fulfillment of the Requirements
for the Degree of Bachelor of Science Magna Cum Laude

EXPERIMENTAL IMPROVEMENTS TO EVENT-TRIGGERED INDIRECT HERDING CONTROL OF A COOPERATIVE AGENT

By

James W. Cross

May 2026

Chair: Warren Dixon

Committee Members: Rushikesh Kamalapurkar, Md Jahidul Islam

Major: Aerospace Engineering

In [1] an event-triggered control method was developed to enable a robotic agent to collaborate with agnostic agents in a multi-agent network. This result is described as herding because the one controlled agent exploits an influence function to indirectly control (herd) agents in the network towards goal locations that only the herder knows. This thesis extends the work of [1] by introducing three modifications designed to yield multi-target experimentation with uncertain target dynamics. The gap this thesis aims to address is how existing event triggered control herding studies lack experimental validation under stochastic drift and multi-target scenarios. A random walk drift model is developed to represent bounded uncertain target motion. Multi-target support is added with sequential switching, where multiple targets are treated as single target problems completed in series. A safety node is written in ROS2 to prevent collisions with the environment during indoor laboratory operations. Experimental validation with one herding agent and three targets is conducted to demonstrate successful regulation of all targets to their respective goals. Although effective, future work will investigate influencing more than one target at a time and reducing the frequency of influence events. These contributions will have broader impacts on improving adversarial target tracking in uncertain environments.

CHAPTER 1 LITERATURE REVIEW

Pursuit-evasion problems are comprised of at least one pursuer agent seeking to intercept at least one target agent [1, 2]. In traditional pursuit-evader problems, the target is antagonistic to the pursuer and has the goal of evading capture. More generally, target agents could also be cooperative or agnostic to the pursuing agent. Indirect control is a broad class of control techniques commonly applied to pursuit-evasion problems which seek to influence target behavior without direct manipulation. Herding is a form of indirect control which utilizes an inter-agent influence model to enable one agent to control another agent. Distance-based influence functions are used in [3, 4] to replicate pursuers entering a limited perception radius around the target agents.

Taking inspiration from biological systems such as sheepdogs managing flocks of sheep, the subject of herding has been of interest for application to target tracking, crowd control, and reduced dependence on high-end capabilities for multi-agent missions [1, 4]. Herding presents a control problem that has a variety of approaches which seek to intercept and eventually indirectly regulate a target to a goal location. Control strategies generally focus on rule-based methods, learning-based methods, and control theoretical methods [1].

Rule-based methods use a small number of simple positioning rules, including using a boid model [5], or involve a greater number of rules specific to the pursuer or target [6]. Additionally, many rule-based methods assume cooperative targets which cannot evade [7] or require a greater number of pursuers than targets [4]. Although some rule-based sources provide stability proofs for extremely limited herding problems [8], rule-based methods are limited to small flock sizes [1] due to insufficient stability guarantees and multiple pursuer requirements. Learning-based methods apply machine learning to learn interaction dynamics or how to apply a learning based method [9]. With similar results to heuristic methods [1], learning-based methods are also limited to small flock sizes because of poor generalization with increasing flock size [9].

The limitations of rule-based and learning-based methods motivated the introduction of control theoretical methods, which use feedback control strategies to regulate target position to a setpoint [1]. Specifically, they offer stability guarantees and support for many targets per pursuer

[4]. Event triggered control (ETC) allows feedback controllers to provide updates in non-uniform time intervals [10], which offers energy and network utilization advantages. ETC checks when a condition is met to trigger an update. The time between consecutive updates is called the inter-event time (IET) and is required to show that the system does not require infinitely many transitions in finite time [1, 10].

An event-triggered indirect herding control framework was introduced in [1] to allow one pursuer agent to regulate one target agent to desired goal locations. A key motivation of the work was to develop a herding controller to enable a single underwater robot equipped with navigation hardware to direct a flock of underwater robots not equipped with navigation hardware to a goal location. This objective lends itself naturally to the pursuit-evader problem because specialized agents can handle navigation for the flock and free up resources on target agents for other mission payloads.

However, conducting multi-agent herding experiments pose coordination, control, and safety challenges because a heterogeneous mix of aerial and ground robots have differing dynamic models. Inter-agent communication and localization are required to maintain synchronization across all agents. Lower-level heading and velocity controllers must also be reliable to form a basis for high-level planning and coordination.

This thesis aims to address experimental improvements to the ETC herding experiment conducted by [1]. Wall collision avoidance, drift dynamics, and multi-agent switching are added to better explore how the control theoretical herding approach in [1] scales to an increasing number of targets. An experiment is conducted using Robot Operating System 2 (ROS2) to communicate with an aerial pursuer and wheeled robot target to determine the impact of these contributions.

CHAPTER 2 PRIOR WORK

Deploying multi-agent controllers onto physical robots requires robust software to handle state estimation, communication, and safety. Prior to developing ETC herding algorithms presented in this thesis, foundational work was conducted to build a multi-agent robot toolkit at the University of Florida (UF) Autonomy Park.

The Autonomy Park is an outdoor laboratory located adjacent to the UF campus in Gainesville, Florida. The Autonomy Park is established with a goal to support the development of multi-agent robotics, nonlinear controls research, robot perception research, and inter-agent communication research in real world conditions. The Autonomy Park consists of a 75x27x18 meter netted enclosure that contains a small hill and grassy field. Additionally the Autonomy Park has two buildings to store robots and run experiments. An aerial view of the Autonomy Park is provided in Figure 2-1. All robots are equipped with RTK-GPS receivers for positioning data, IMUs for orientation data, and are running ROS2 Humble to network and distribute state information and execute controller outputs.

Figure 2-1. Aerial view of the Autonomy Park netted enclosure.

2.1 Technical Challenges and Solutions

2.1.1 Localization Challenges

The Autonomy Park facilitates multi-agent experiments with the option for over 20 vehicles being independently controlled simultaneously. Accurate localization is necessary to evaluate controller performance and safely control multiple robots in a confined area. Inaccurate sensors

may cause reported robot positions to shift erratically by greater than several body lengths, even if the robot is not moving. Collision avoidance and potential field boundary programs are used to minimize the risk of a collision, which both rely on accurate position estimates to correct hazardous trajectories. Accurate position estimates are critical for robotics and especially for outdoor robotics where nonlinearities caused by the environment can influence an experiment.

2.1.2 Obtaining Latitude and Longitude

An RTK-GPS system has been successfully used for accurate localization within the Autonomy Park. Uneven heating of the troposphere and charged particles from the sun in the ionosphere alter the speed of signals from GPS satellites. The result is that for a given location, the reported value from a GPS receiver will vary. RTK-GPS mitigates this error because the technology consists of a base station located at a fixed location broadcasting measurement corrections to compatible receivers mounted on the robots. At the Autonomy Park, an Emlid Reach RS3 RTK base station is attached to a tower adjacent to the netted enclosure. An accurate position reading was obtained by recording approximately 24 hours of GPS data with the receiver attached to the tower and processing the data using GPS Precise Point Positioning.

2.1.3 Maintaining Altitude

One consequence of the Autonomy Park's north Florida location is that summer storms can form and move inland over the course of a day. The aerial vehicles at the Autonomy Park rely primarily on barometric pressure to determine altitude. A consequence of rapidly developing summer storms is that atmospheric pressure at ground level can vary by up to 3 kPa. During these events, multirotor aircraft have been observed accelerating upwards into the netted enclosure while attempting to hold a stable position. The altitude discrepancy is believed to be the result of a sufficient difference in the AMSL pressure of the region with the actual local barometric pressure.

Solutions to this problem are ongoing and include a LIDAR coupled with a height map and meteorological equipment to offer corrections. Single beam LIDARs have been attached to the multirotor aircraft and pointed directly downwards to measure the distance to the ground. Onboard flight computers are set to maintain a fixed height above the ground. The main limitation with

this approach is that aerial robots cannot fly over ground vehicles or other aerial robots because it would result in erroneous distance readings from the LIDAR beam contacting the robot instead of the ground. Additionally, altitude estimates lose accuracy when the vehicle is pitching forward and the LIDAR beam is no longer pointed directly downwards.

2.2 Software Framework Design

The presented software toolkit serves to aggregate and fuse sensor data to generate state information for the intended robot platform, in a common reference frame among all robots. The state information is used by the experimental controller to process the control outputs. The product is a software suite that facilitates the input of state data to controllers and handles the output of velocity messages using the same data types and overall code structure in simulation and on various robots. More specifically, ground robots with fewer degrees of control than aerial robots should be able to receive the same commands from a controller.

ROS2 is a middle-ware that runs the Autonomy Park robots allowing a single node to interface with multiple robots. ROS2 uses a network of topics to distributively share information, eliminating the need for a central computer to send individual commands to every robot. The software toolkit is comprised of a Python simulator, ROS2 simulator, and ROS2 experiment package.

2.2.1 Core Components

The Python simulation is designed to allow researchers to convert controllers written in other languages to Python code that uses the similar variable names and program structure to common ROS2 nodes. A matplotlib window is included to visualize trajectories calculated by integrating the controller output using a common Euler integration method. The Python simulation is not intended to replace tools used for controller evaluation, but to verify correct implementation in a format that can be easily copied into ROS2. The ROS2 simulation runs the same code from the Python simulation and verifies implementation in a ROS2 environment. An RVIZ visualization, in addition to a 2d plot, is generated when the controller is running. For the herding example, the visualization is composed of herding agent and target agent nodes along with an integrator and visualization node. The nodes include common functions that influence dynamics, behavior

and evasion dynamics to customize the characteristics of each robot. The ROS2 experiment code modifies the ROS2 simulation package by removing integrator nodes and routing output velocity commands directly to the robots. Robot position estimates are drawn from the fused state data and passed into the controller. Additionally, key values including target agent tracking error, position, and velocity are saved to a CSV file for post-experiment analysis.

2.2.2 Functions

A number of functions are included to aid in controller implementation. The software toolkit was designed to initially support a herding experiment which consists of a herding agent directing a target agent towards a goal location. The herding agent and target agent nodes contain relevant functions, are subscribed to other robot positions, and can publish commands to the onboard motor driver.

The wheeled ground vehicles at the Autonomy Park are either mechanically differential-drive robots or lack individually addressable motor control. Effectively, all the wheeled ground vehicles can accept only a forward or backward velocity and an angular velocity about the vertical axis. The `convert_velocity` function converts non-holonomic velocity commands from a controller into linear and angular velocities for the ground vehicles. A Euclidean norm and atan2 of the controller output velocity are used.

The primary benefit to this approach is that researchers can treat each robot as interchangeable agents, sending the same velocity messages to quadcopters and differential drive robots. This approach effectively treats ground robots as holonomic agents in a heterogeneous experiment. Limitations of this approach include how the software toolkit currently only works with 2d simulations, i.e. the quadcopters do not change altitude and all robot positions are projected onto a plane parallel to the ground. Additionally, the program is limited to controllers that output velocity commands. Because of this limitation, 2nd order control is not directly available.

Distance is calculated using the Euclidean norm of the positions of each robot. The distance function is calculated within the Autonomy Park frame. The Autonomy Park frame is defined as a Cartesian space with the origin centered in the middle of the park on the ground. To accomplish

a simplified coordinate space in the real world, the ROS2 transform (TF) module was used to transform reference frames and convert sensor data onto the centroid of the robots and ultimately express the position and orientation of each robot to the geometric center of the Autonomy Park. The use of a common reference frame simplifies experiment design, because goal locations and expected trajectories are expressed in XYZ values. The robots natively receive position information from RTK-GPS receivers, but then the data is fused with IMU and odometry data using the Robot Localization ROS2 package. The TF module is used within the robot localization package to handle the transformations to the Autonomy Park reference frame.

The controller function contains the controller being tested and is provided with every robot's position and outputs velocity commands either directly to a robot or to `convert_velocity` first. By running a separate node for every agent, the controller is able to run onboard each robot. Onboard processing is critical to several types of planned experiments which incorporate jamming and uncertain state information, where the robot must still be able to execute its mission without a connection to a central computer.

The `interaction_function` is critical to planned herding experiments where target agents are herded towards a goal location using some interaction dynamic. The function is currently designed to cause the target agent to move in the opposite direction of the herding agent when the distance function is below a given threshold. Future improvements include adding uncertainty into the target agent response.

Additionally, it may be desirable for target agents in a herding experiment to move in a general area when not being actively herded. Loitering is controlled with the `levy_walk` function, which uses a Pareto distribution to output small linear and angular velocities, such behavior is analogous to animals in the real world that may wander when left unattended.

CHAPTER 3 CONTROL IMPROVEMENTS

Applying lessons learned from prior work, three modifications are made to the original framework developed in [1]. The modifications include introducing stochastic target drift, discrete switching logic for multi-target regulation, and a velocity-based safety filter for laboratory experiments.

3.1 Herding Framework

The following event-triggered herding framework was developed in [1]. The original work tasks a single pursuer agent with regulating a single target agent to a known goal location, $\zeta_g \in \mathbb{R}^n$. Target position is denoted with $\eta_t(t) \in \mathbb{R}^n$, and target dynamics are:

$$\dot{\eta}_t(t) \triangleq f(\eta_t(t)) + \mu_t(\eta_t(t_k), \eta_p(t_k)), \tag{3-1}$$

where $f(\eta_t(t))$ represents the target's drift dynamics, which are assumed to be continuously differentiable and bounded with $\|f(\eta_t(t))\| \leq \bar{f}$. The term $\mu_t(\eta_t(t_k), \eta_p(t_k))$ is the influence applied by the pursuer in discrete time steps t_k. This influence incorporates a zero-order hold function with:

$$\mu_t(\eta_t(t_k), \eta_p(t_k)) \triangleq \begin{cases} \nu_t \left(\frac{\eta_t(t_k) - \eta_p(t_k)}{\|\eta_t(t_k) - \eta_p(t_k)\|} \right), & \text{if } \|\eta_t(t_k) - \eta_p(t_k)\| \leq R_p \\ 0, & \text{otherwise,} \end{cases} \tag{3-2}$$

where $\nu_t \in \mathbb{R}_{>0}$ is the constant speed of the target, and $R_p \in \mathbb{R}_{>0}$ is the maximum influence range of the pursuer. The pursuer agent has position defined with $\eta_p(t) \in \mathbb{R}^n$ and directly controllable dynamics:

$$\dot{\eta}_p(t) \triangleq \mu_p(t). \tag{3-3}$$

The pursuer aims to track a desired trajectory defined as

$$\eta_d(t) \triangleq \eta_t(t) + R_a S_0 \tag{3-4}$$

where R_a is the desired influence range and S_0, defined as $S_0 \triangleq e_t(t)/||e_t(t)||$, denotes the unit vector pointing from the target to the goal location.

The target position error is defined as $e_t(t) \triangleq \eta_t(t) - \zeta_g$, and the pursuer tracking error is defined as $e_p(t) \triangleq \eta_d(t) - \eta_p(t)$. The pursuer agent controller is:

$$\mu_p(t) \triangleq \dot{\eta}_d(t) + (\beta + 1)e_p(t), \tag{3-5}$$

where $\beta \in \mathbb{R}_{>0}$ is a user-selectable control gain.

To reduce energy usage required for continuous control, the pursuer agent only influences the target μ_t when indicated by a trigger function T. The IET function is defined as $\tau \triangleq t - t_k$, and used to track the time since the last update at t_k.

To prevent the system from encountering Zeno behavior, infinite control commands in finite time intervals, the trigger function evaluates the system states against a user-defined minimum IET ($\delta_{\min}$) and a theoretically derived maximum IET ($\delta_{\max}$):

$$T \triangleq \begin{cases} \gamma_s, & 0 \leq \tau < \delta_{\min} \\ \gamma_s + \|e_p(t)\|^2 - (\alpha + 1)\|e_t(t)\|^2, & \delta_{\min} \leq \tau < \delta_{\max} \\ 0, & \text{otherwise,} \end{cases} \tag{3-6}$$

where $\alpha \in \mathbb{R}_{>0}$ is a user-selected control gain, and γ_s is an adaptive trigger threshold defined as $\gamma_s \triangleq \frac{1}{2}\bar{f}^2 + \frac{1}{2}\nu_t^2 + (\alpha + 1)\|e_t(t_k)\|^2$.

3.2 Stochastic Target Drift

While the original work assumes the target agent's drift dynamics, $f(\eta_t(t))$, are governed by $f(\eta_t(t)) = \tanh(\eta_t(t))$ [1], target behavior can be unpredictable. Targets may enter into holding patterns to search for the pursuer, or may encounter obstacles that cause erratic disturbances. To more accurately evaluate controller performance, the target position, $\eta_t(t)$, is subjected to a discrete-time random walk [11]. Despite its randomness and nonlinear output, a random walk was selected because of its statistical upper bound, used in $\bar{f}$ to calculate the maximum IET. Specifically, the

uniform distribution is bounded on $[0, 2\pi]$ to select a heading, while the target velocity uses a Pareto distribution. Although a Pareto distribution can have infinite upper tail growth, practical physical limits of the robot acceleration and maximum velocity constitute an upper bound. In simulation, the forward velocity is capped by the physical constraints of the experimental robot platform.

The step vector $\Delta\eta_t(t_k)$ is defined by a step size s_k and a heading angle θ_k and added to the current target position $\eta_t(t_k)$:

$$\eta_t(t_{k+1}) = \eta_t(t_k) + \Delta\eta_t(t_k) \tag{3-7}$$

$$\Delta\eta_t(t_k) = \begin{bmatrix} s_k \cos\theta_k \\ s_k \sin\theta_k \end{bmatrix}. \tag{3-8}$$

The step size and heading angle are drawn from the following statistical distributions:

$$s_k \sim \text{Pareto}(\alpha_{Pareto}), \quad \theta_k \sim \text{Uniform}(0, 2\pi). \tag{3-9}$$

Additionally, a communication radius is used to determine if the pursuer can influence the target. Physical systems have limited sensing ranges and the communication radius simulates the impact of the pursuer just coming into view of the target. The pursuer retains total state information and only the targets are impacted by this radius.

In the original work [1], the magnitude of influence applied to the target, $|\mu_t|$, is set at ν_t, the target's constant velocity. Effectively, [1] assumes constant target motion and only redirects the target using the influence. Because the random walk can have velocities that vary based on the Pareto distribution, when the pursuer is within the communication radius of the target, the target velocity is set to the value of ν_t. This maintains the random walk that causes targets to drift unherded at various speeds, and satisfies the assumptions of [1] used to calculate $\bar{f}$ while actively pursued. Physically, this supports the idea that when a target can see its pursuer, it will attempt to evade at its maximum speed. The angular disturbances from the uniform distribution remain while being actively herded and result in an uncooperative target. A uniform distribution was selected to ensure that the target has no preferred direction when free to drift.

3.3 Multi-Target Switching Logic

The framework in [1] was designed for one pursuer and one target. To handle multiple targets, a switching rule was developed to select the target with greatest goal error and focus all pursuer influence exclusively on the active target.

Once the active target is regulated to a tolerance around the goal location ε, the pursuer switches to the next target that is furthest from its goal. The sequential approach treats the multi-target case as a series of one-on-one herding events in succession, maintaining the stability guarantees derived in the single-agent case. Global error remains practically bounded because the pursuer does not switch until the target error is bounded by ε, and the unherded targets error is bounded by the physical limits of the random walk.

3.4 Experimental Safety Filter

To ensure safe operation of the Parrot Bebop quadcopter within the indoor laboratory bounds, a safety filter is applied to the control input $\mu_p(t)$ [1]. The filter generates a repulsive velocity command, $\phi(d)$, based on the distance, d, to the nearest boundary. Shaping parameters a and b are combined with a transition distance p to specify how the repulsive magnitude changes with d

$$\phi(d) = \begin{cases} ad^{-b}, & 0 < d \leq p \\ -abp^{-(b+1)}d + ap^{-b}(1+b), & p < d \leq \rho_{\min} \\ 0, & d > \rho_{\min} \end{cases} \tag{3-10}$$

$$\rho_{\min} = p\frac{1+b}{b} \tag{3-11}$$

where $\rho_{\min}$ is the activation distance. This safety filter applies zero influence when the agent is outside the activation distance, weak influence that scales linearly with the transition distance, and strong polynomial influence when very close to the wall. The safety filter runs in its own ROS2 node and receives messages on the `/cmd_vel_unfiltered` topic that are filtered and retransmitted to the agents on `/cmd_vel_filtered`.

CHAPTER 4 SIMULATION

For the numerical results in Figures 4-1 to 4-6, the following parameters were selected to compare results with [1] where applicable: $\alpha = 2.1$, $\beta = 5.0$, $R_a = 5.0$, $R_p = 10.0$, $\nu_t = 1.5$, $\delta_{\min} = 0.1 \cdot \delta_{\max}$. An $\alpha_{Pareto} = 4.0$ was used (with the `numpy` library for statistical distributions using a seed value of 23) and the resulting γ_u was calculated to be $\gamma_u = 2.25$ from [1]. When targets reach their goal locations, they are configured to stop moving. Additionally, a rectangle is overlaid to indicate the safe bounds of the indoor experimentation facility.

4.1 Single Pursuer Single Target

Building on the single-pursuer single-target simulation from [1], Figure 4-1 demonstrates the trajectory with random walk drift. The $\delta_{\max}$ was found to be $\delta_{\max} = 0.1178$ based on the initial conditions in Figure 4-1. Compared to [1], the target error decreases linearly in Figure 4-2 with jumps before trigger events from the random walk activating when the target exits influence range. In Figure 4-3, the trigger function magnitude after the initial influence event gradually trends downwards, representing effective control of the target when approaching the goal location.

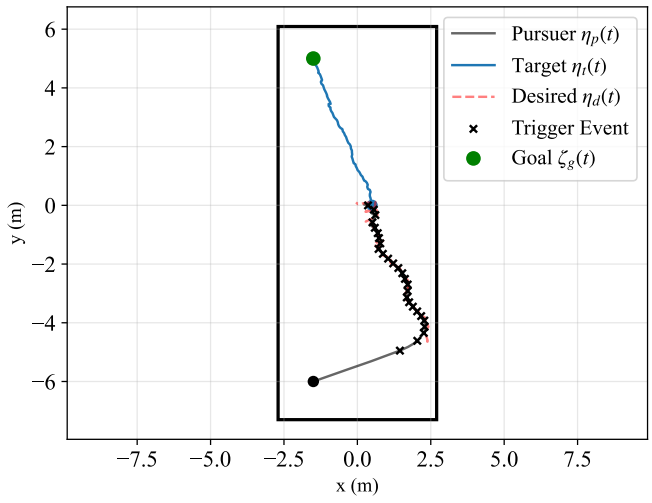

Figure 4-1. Trajectories of the pursuer agent, $\eta_p(t)$, and target agent, $\eta_t(t)$, are represented by the black and blue lines, respectively. The green circle represents the goal location, $\zeta_g(t)$ while the black and blue circles represent the starting locations of the pursuer, $\eta_p(t_0)$, and target, $\eta_t(t_0)$. The black ×'s represent trigger events, when the pursuer applies an influence to the target.

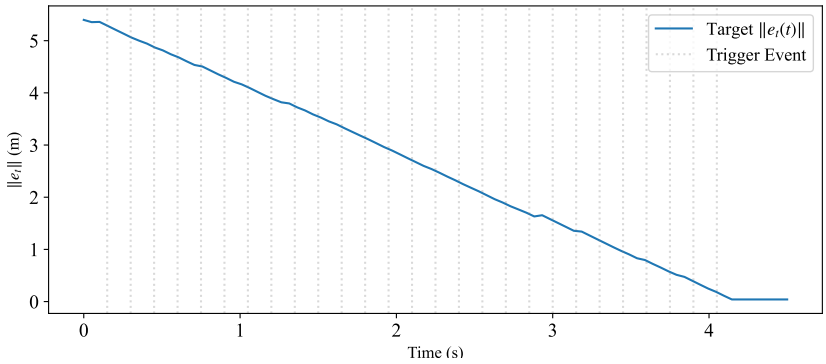

Figure 4-2. The magnitude of target goal error, $||e_t(t)||$, over time. Vertical dashed lines indicate trigger events, when the target is influenced by the pursuer.

The trigger event spacing in Figures 4-2 and 4-5 appears uniform due to the minimum and maximum IET limits, $\delta_{\min}$ and the $\delta_{\max}$. The trigger function saturates within these analytical bounds to cause periodic-like updates. The wide spacing of the trigger events indicates that the system is being limited by $\delta_{\max}$, triggering an update to remain in the stable subsystem. The value of $\delta_{\max}$ is a result of the stability analysis in [1] and is a function of initial error bounds and control gains. The trigger function allows for increased update frequency if the target is not tracking the desired trajectory before $\tau = \delta_{max}$, but will default to the timer to guarantee asymptotic stability.

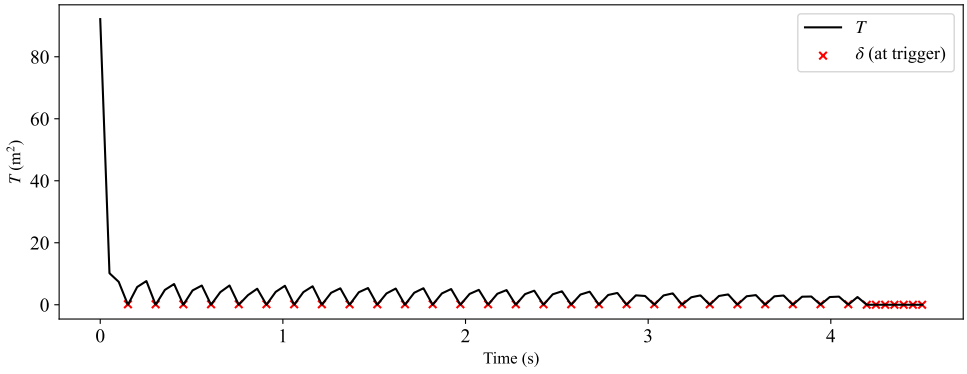

Figure 4-3. Trigger function value, T, over time with trigger events denoted by δ. The initial spike represents the decision to have the pursuer immediately begin regulating the target.

4.2 Single Pursuer Multiple Targets

The algorithm from [1] was adapted to support multiple agents through sequential switching and evaluated with random walk disturbances. Figure 4-4 demonstrates the trajectory with random

walk drift. Because the multi-target approach treats the problem as sequential single targets and based on the initial conditions in Figure 4-4, the $\delta_{\max}$ was also found to be $\delta_{\max} = 0.1178$. In Figure 4-5, target error decreases approximately linearly when actively herded, and the target remains in the proximity of their initial position from the drift dynamics. In Figure 4-6, the trigger function magnitude after the initial influence event when the target is marked active gradually trends downwards, representing effective control of the target while approaching the goal location.

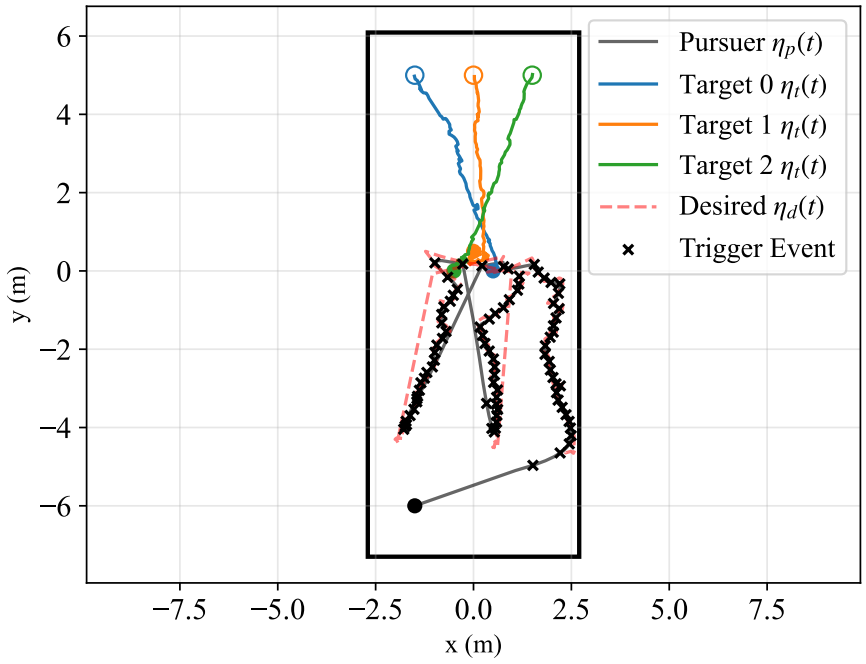

Figure 4-4. Trajectories of the pursuer agent, $\eta_p(t)$, are represented by the black line, while each target agent, $\eta_t(t)$, is represented by its own color. The open circles correspond to each target's goal location, $\zeta_g(t)$ while solid circles represent the starting locations of the pursuer, $\eta_p(t_0)$, and targets, $\eta_t(t_0)$. The black ×'s represent trigger events, when the pursuer applies an influence to the active target.

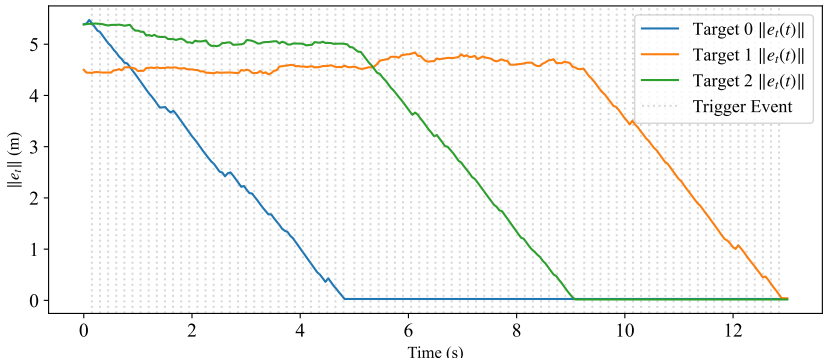

Figure 4-5. The magnitude of each target's goal error, $||e_t(t)||$, over time. Vertical dashed lines indicate trigger events, when the active target is influenced by the pursuer.

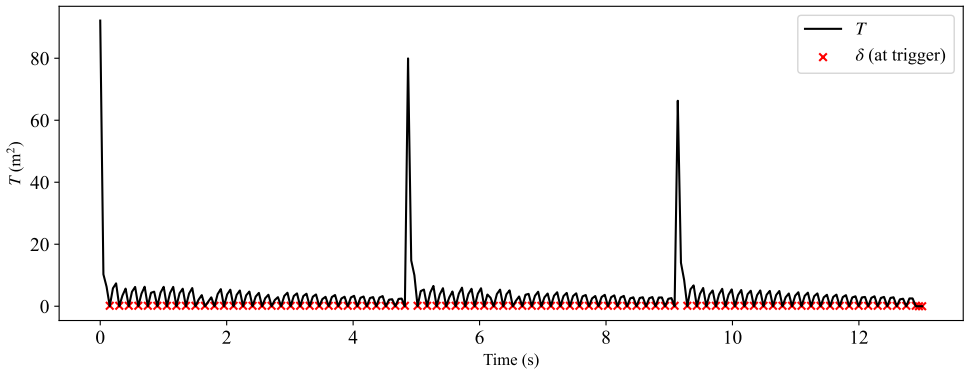

Figure 4-6. Trigger function value, T, over time with trigger events denoted by δ. The large spikes correspond to the pursuer starting to herd a target and exist because high initial T results in immediate influence at the start.

CHAPTER 5 EXPERIMENTATION

This section presents the results of several experiments conducted to determine herding performance with random walk drift dynamics and multiple targets. Much of the software work completed to support multi-agent experiments at the Autonomy Park was applied in configuring this experiment which used an indoor laboratory.

5.1 Safety Calibration

The safety node was calibrated by selecting 4 points in the world frame to define a safe boundary at the corners of the experimental area of the indoor laboratory: $(-2.7, -7.3)$, $(2.7, -7.3)$, $(2.7, 6.09)$, $(-2.7, 6.09)$. The available space was approximately 13×5 m^2.

Safety parameters were initially selected by plotting Eq. (3-10) with values chosen to place the activation distance approximately 1 meter from the wall. Variables a and b were selected to place the location where the linear section transitions to the stronger polynomial section approximately 0.8 meters from the wall. The final safety parameters were: $a = 0.005$, $b = 4.1$, and $p = 0.8$. The corresponding influence function, Eq. (3-10), is plotted in Figure 5-1. Manual control of a Parrot Bebop quadcopter confirmed that the aircraft stopped at the safe boundary and drifted away from the wall when velocity commands ceased.

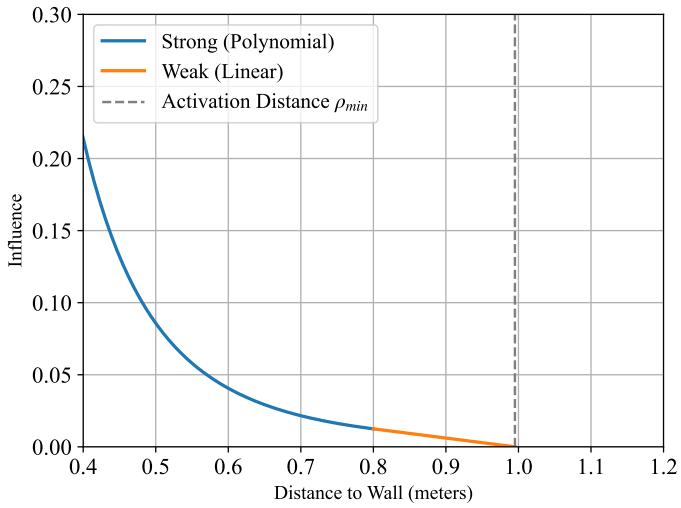

Figure 5-1. Zoomed influence magnitude with distance to wall, d.

5.2 Experiment Configuration

An experiment was conducted to analyze the impact of stochastic drift and sequential switching. A Parrot Bebop 2.0 quadcopter and three Husarion Rosbot 2 Pro differential drive ground robots are used for the pursuer and three targets, respectively. The robots are shown in Figure 5-2 and a snapshot of the experiment is presented in Figure 5-3.

Figure 5-2. Husarion Rosbot 2 Pro (left) and Parrot Bebop 2.0 (right). At least 4 motion capture markers are attached to each robot for tracking.

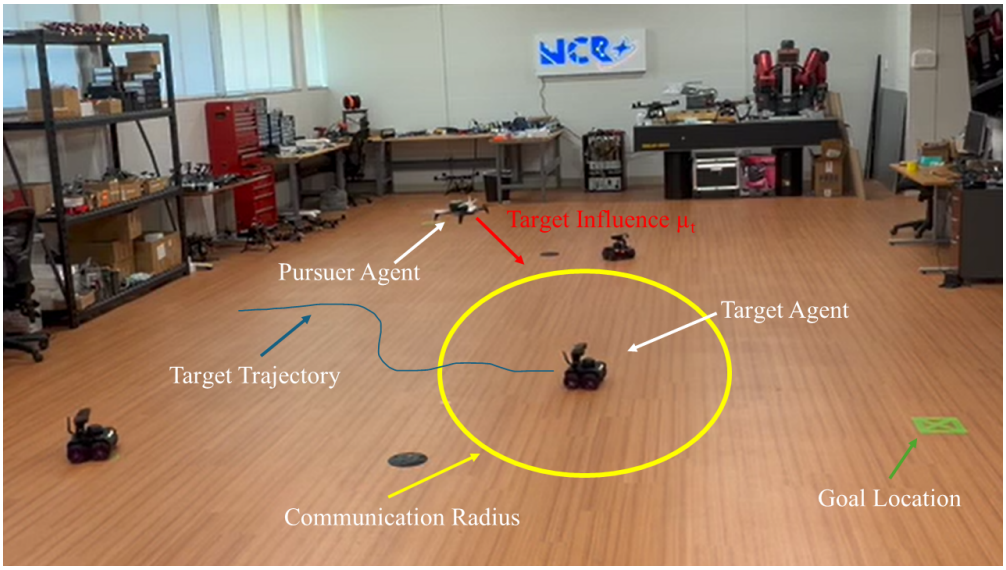

Figure 5-3. Still frame from experiment video recording. The pursuer is airborne and actively herding the target with greatest goal error (rosbot05). Targets not actively herded are free to move according to stochastic drift, unless their goal has been reached.

The Rosbot is equipped with wheel encoders and an IMU to regulate low-level forward and angular velocity. The low-level control loop runs on a Husarion CORE2 board which communicates

with the main *UP board* computer using micro-ROS. The main Rosbot computer runs Ubuntu 22.04 with ROS2 Humble to communicate with the experiment computer over WiFi on an enclave network. The manufacturer software support for these devices is poor and configuration of the onboard Ubuntu firewall was necessary to prevent the default micro-ROS configuration from broadcasting on topics without the correct namespace. The only topic exposed on each Rosbots was a namespaced `cmd_vel` topic which listened for a linear x command and an angular z command and is shown in Figure 5-4.

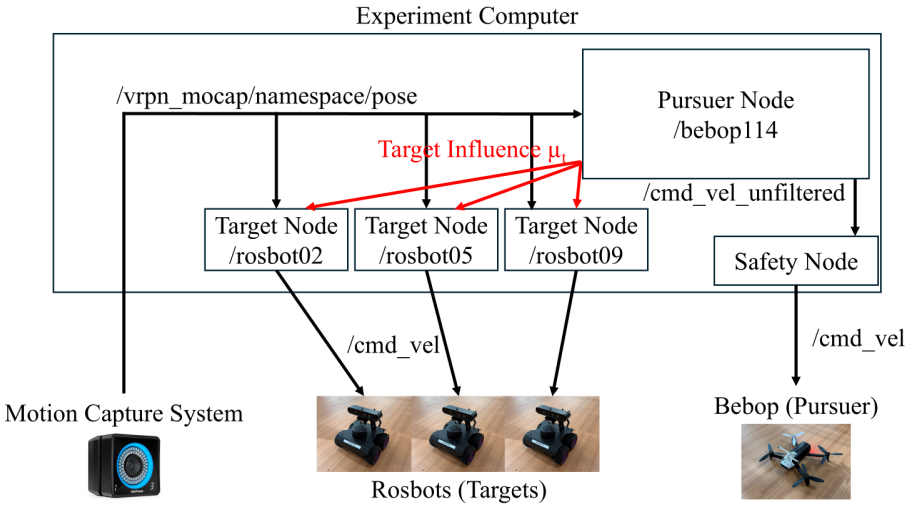

Figure 5-4. Experimental ROS2 architecture. The pursuer can access all target poses and each target is provided its own pose as the heading controller requires orientation information. Pursuer commands are filtered through the safety node to prevent the Bebop from exiting the bounds of the experiment.

The Parrot Bebop 2.0 quadcopter was configured to receive filtered velocity commands from the experiment computer over the network. An experiment computer running ROS2 Humble and Ubuntu 22.04 ran the main herding node in addition to the safety node, data logging, quadcopter communication, and motion capture updates. Position information was provided by a NaturalPoint, Inc. OptiTrack motion capture system at 120 Hz. Linear and angular position standard deviations are approximately 0.01m and 1°.

Due to the nonholonomic dynamics of the Rosbot, a proportional (P) controller, where the control input is proportional to the tracking error, is used to align the heading of each target with the control command. Similarly, a P controller was used on the Bebop to align its heading with the target to more clearly show what target is being actively herded. A P controller is also used to regulate the Bebop altitude to a setpoint of 1 meter off the ground because aggressive maneuvering may result in a loss of lift. Target configuration parameters, provided in Table 5-1, were constant across all Rosbots, with the exception of independent goal locations.

Table 5-1. Experiment Configuration Parameters for Agent and Target Nodes

Description	**Variable Name**	**Value**
Time Step	`dt`	0.005 s
Target Velocity	v_t	0.25 m/s
Goal Radius	ε	0.4 m
Target Sight Range	R_p	1.5 m
Desired Influence Radius	R_a	0.5 m
User Selected Control Gain	α	2.1
User Selected Control Gain	β	4.0
Pursuer Maximum Speed	v_{max}	1.0 m/s
Goal Coordinates (x, z)		Varied[b]

[a] Targets used in the experiment were: `rosbot09`, `rosbot05`, `rosbot02`. Pursuer used was `bebop114`.
[b] Assigned respective goals at $(1.434, -5.296)$, $(-2.418, -2.209)$, and $(1.696, 5.207)$.

Additionally, an $\bar{f} = 4.0$ was conservatively estimated based on using a Pareto distribution for the drift step size with an $\alpha_{Pareto} = 3.0$.

5.3 Experimental Results

Figure 5-5 depicts the trajectories of the agents along with goal locations and switching events. The pursuer begins at the center of the laboratory and initially looks for the target with greatest goal error. After selecting rosbot05, the traget is regulated to its goal location. A goal radius was used to ensure that the target is considered successfully regulated once it reaches a neighborhood around the goal location. Due to the onboard velocity controller performance, targets do not immediately stop at the edge of the goal tolerance.

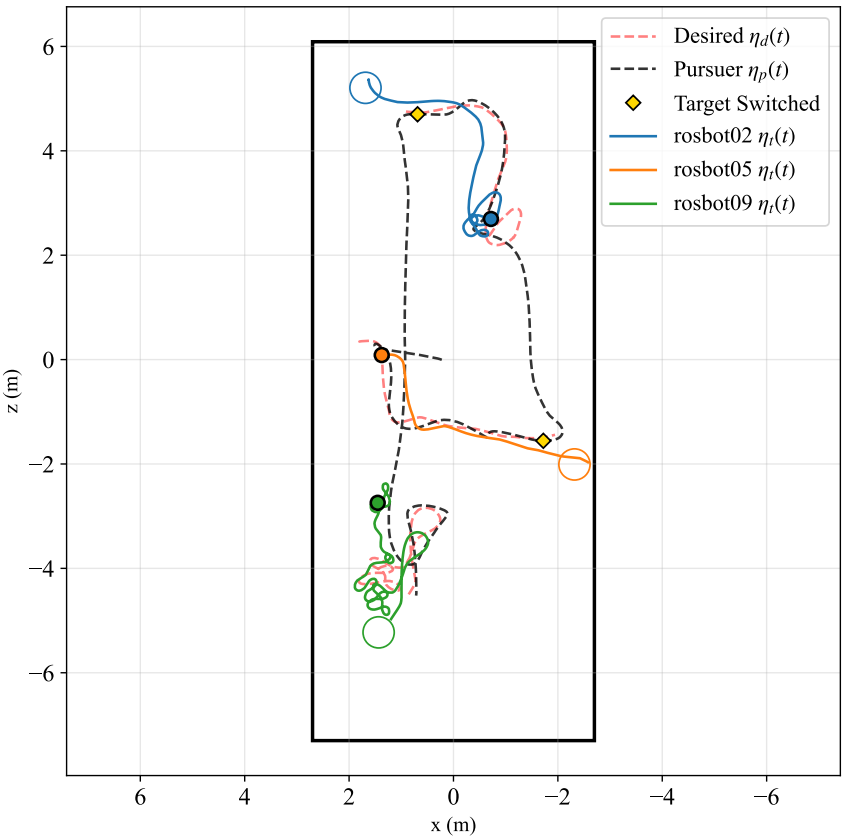

Figure 5-5. Trajectories of the pursuer agent, $\eta_p(t)$, are represented by the dashed black line, while each target agent, $\eta_t(t)$, is represented by its own color. The open circles correspond to each target's goal location and goal radius ε, $\zeta_g(t)$ while solid circles represent the starting locations of the pursuer, $\eta_p(t_0)$, and targets, $\eta_t(t_0)$. The yellow diamonds indicate when switching events occur, and precede sections where the pursuer is in transit to the next target. Influence markers were omitted in the plot due to small IETs.

The impact of switching is clearly visible in Figure 5-6 where the initial target, rosbot05, is regulated to its goal location while rosbot02 and rosbot09 are free to drift around the environment. Target goal overshoot shown in Figure 5-5 also is visible in 5-6, because the target goal error continues to change for a brief period after the pursuer switches to the next target. Additionally, the drift trajectory of rosbot09 causes it to temporarily approach the goal location despite having no preferred drift direction. Rosbot09 experiences an increase in target goal error near the end of being actively herded due to the drift dynamics having a comparable magnitude to the pursuer influence. All targets were successfully regulated within a specified goal radius, demonstrating the robustness of sequential switching and the original herding controller in [1] to stochastic disturbances.

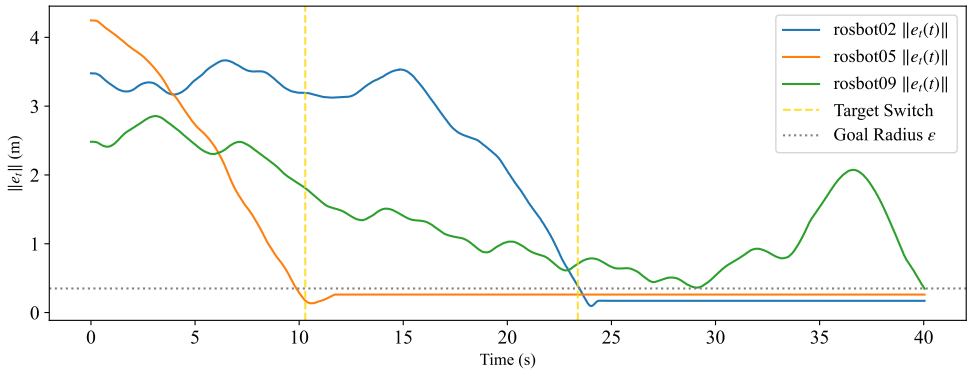

Figure 5-6. The magnitude of each target's goal error, $||e_t(t)||$, over time. The goal radius ε is plotted and the experiment was designed to conclude once all targets could be found within the goal radius of their goal locations. Target switching events are indicated in yellow. Trigger events were omitted in the plot due to small IETs.

Experimental results demonstrate the effectiveness of the controller and trigger function in [1]. The evolution of the trigger function value in Figure 5-7 clearly shows how sequential switching resets T when moving to the next target. The impact of stochastic drift causes an increase in trigger values because the random walk opposes the target influence and the controller demands more rapid influences to compensate. A limitation of the current work is the small IET value which results in frequent influence events. Future work could consider how to reduce influence frequency to save pursuer energy.

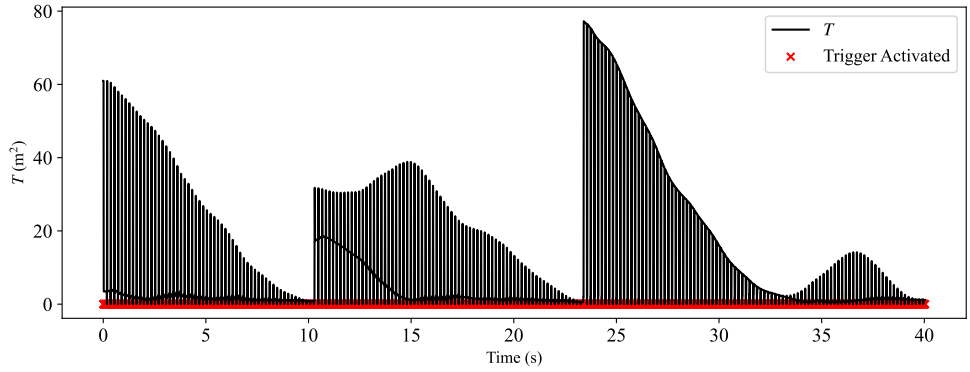

Figure 5-7. Trigger function value, T, over time with trigger events denoted by δ. The large spikes correspond to the pursuer starting to herd a target and exist because high initial T results in immediate influence at the start.

CHAPTER 6 CONCLUSION

This thesis addresses three modifications made to [1] which are focused around replicating the experiment in an indoor environment and adding multi-target support with a different drift dynamic. A random walk drift was added to replicate uncertain drift dynamics, because it relies on statistical distributions to provide bounded uncertainty. Multi-target switching was accomplished by sequentially regulating a single target to its goal then switching. Finally, a safety node in ROS2 was written to prevent the pursuer from colliding with the walls. An experiment was conducted with three targets and one pursuer and resulted in successful regulation of all targets to their goal locations. Although functional, a limitation of the current switching approach is how influence can only be applied to a single target, even if multiple are in proximity. Future work could consider how stability guarantees could be maintained when the pursuer influences all targets in range, simultaneously. Broader impacts include applications to adversarial target tracking in uncertain environments.

LIST OF REFERENCES

[1] P. M. Amy, B. C. Fallin, J. N. Philor, and W. E. Dixon, "Event-triggered indirect herding control of a cooperative agent," *IEEE Robotics and Automation Letters*, vol. 11, no. 3, pp. 3828–3835, 2026.

[2] S. D. Bopardikar, F. Bullo, and J. P. Hespanha, "On discrete-time pursuit-evasion games with sensing limitations," *IEEE Transactions on Robotics*, vol. 24, no. 6, pp. 1429–1439, 2008.

[3] R. A. Licitra, Z. I. Bell, E. A. Doucette, and W. E. Dixon, "Single agent indirect herding of multiple targets: A switched adaptive control approach," *IEEE Control Systems Letters*, vol. 2, no. 1, pp. 127–132, 2017.

[4] R. A. Licitra, Z. D. Hutcheson, E. A. Doucette, and W. E. Dixon, "Single agent herding of n-agents: A switched systems approach," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 14 374–14 379, 2017.

[5] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," in *Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, 1987, pp. 25–34.

[6] T. Miki and T. Nakamura, "An effective simple shepherding algorithm suitable for implementation to a multi-mmobile robot system," in *First International Conference on Innovative Computing, Information and Control-Volume I (ICICIC'06)*, vol. 3, 2006, pp. 161–165.

[7] Q. Chen and J. Luh, "Coordination and control of a group of small mobile robots," in *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*. IEEE, 1994, pp. 2315–2320.

[8] B. Pshenichnyi, "Simple pursuit by several objects," *Cybernetics*, vol. 12, no. 3, pp. 484–485, 1976.

[9] N. K. Long, K. Sammut, D. Sgarioto, M. Garratt, and H. A. Abbass, "A comprehensive review of shepherding as a bio-inspired swarm-robotics guidance approach," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 4, no. 4, pp. 523–537, 2020.

[10] W. P. M. H. Heemels, K. H. Johansson, and P. Tabuada, "An introduction to event-triggered and self-triggered control," in *Proceedings of the 51st IEEE Conference on Decision and Control (CDC)*, 2012, pp. 3270–3285.

[11] V. Zaburdaev, S. Denisov, and J. Klafter, "Lévy walks," *Reviews of Modern Physics*, vol. 87, no. 2, pp. 483–530, 2015.