Wildlife Monitoring Using Camera Traps

Miruna Oprescu (amo78)

Abstract

Wildlife monitoring using camera technology has recently emerged as a promising tool for biodiversity conservation efforts. Motivated by the monitoring and study of the dwindling population of Snow Leopards in Central Asia, this technical report details the design and implementation of an automated wildlife monitoring system using camera traps. We model the camera traps as a Pan-Tilt (PT) camera system, capable of automated horizontal panning and vertical tilting, to simulate a field of view (FOV) that captures both stationary and moving objects within a monitored environment. Our system integrates the YOLOv2 [1] neural network for object detection and the SORT [2] algorithm for tracking. It distinguishes between static and dynamic targets using kinematic matching in the inertial frame and employs intelligent camera control to maintain continuous observation of the target within the FOV. Our simulations effectively capture the complex dynamics of the system and achieve a high degree of accuracy in the downstream tasks, thus underscoring the potential of such advanced monitoring techniques in wildlife conservation efforts.

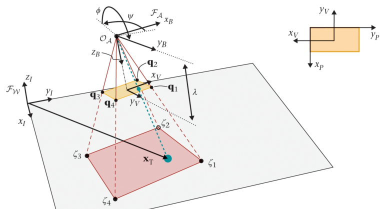
Introduction

Advanced camera technologies, particularly Pan-Tilt (PT) cameras with their automated horizontal panning and vertical tilting capabilities, have revolutionized various fields, including surveillance, video conferencing, television production, and remote sensing. These technologies are particularly promising in the realm of wildlife conservation. By enhancing the quality and reliability of wildlife monitoring, they play a pivotal role in conservation efforts, offering valuable insights through detailed behavioral data on endangered species. This approach not only improves monitoring methodologies but also significantly contributes to the protection and study of species facing extinction.

One notable application in the realm of wildlife conservation is the monitoring and study of the dwindling Snow Leopard (Fig. 1a) population, inspired by the author's work with The Snow Leopard Conservancy. This non-profit organization focuses on studying and monitoring snow leopards within their natural habitat in Central Asia. The current estimated population of Snow Leopards is between 4,000 and 6,600 [3], making them a vulnerable species facing a variety of threats such as as habitat loss due to mining, poaching, retribution killing, loss of prey, etc [4]. The Snow Leopard Conservancy utilizes trail cameras (Fig. 1b), also known as camera traps, along travel routes frequented by snow leopards. These cameras capture digital images and videos, which are analyzed to identify individual snow leopards based on their

(a) Snow leopard image capture from a trail camera.

(b) Camera trap in snow leopard habitat.



(c) PT camera model and FOV geometry.

Figure 1: (a)-(b): Snow leopard in its natural habitat and camera trap, courtesy of The Snow Leopard Conservancy [3]. (c): PT camera FOV and model. Figure sourced from [5].

distinct pelage spot patterns, akin to a form of “fingerprinting” [3]. However, these efforts could be enhanced by incorporating intelligent camera control to optimize objectives such as detecting as many targets as possible or tracking targets across the environment. This would provide a better understanding of Snow Leopards’ behavior, contributing significantly to their conservation.

In this work, we implement a similar scenario where the snow leopards (“targets”) are allowed to roam freely within a monitored workspace populated with static objects (“objects”) and the goal of the camera trap is to either a) detect as many targets as possible within a given time frame, or b) track targets across the simulated workspace. We model the camera trap as a remotely controllable PT camera with adjustable pan and tilt angles (ψ, ϕ) to effectively capture a region of interest (ROI) within the camera’s FOV (Fig. 1c). Camera control is achieved through two applied voltages, u_1 and u_2, that are proportional to pan and tilt angular velocities, $\dot{\psi}$ and $\dot{\phi}$, respectively. The camera’s state, denoted as $\mathbf{s} = [\psi\ \phi\ \dot{\psi}\ \dot{\phi}]$, is typically constrained to ranges determined by physical limitations on the system. In Table 1, we give common values for these ranges: The PT camera surveys a

Table 1: Common parameters of PT cameras.

Description	Variable	Value
Pan angle	ψ	$[0, 2\pi)$
Tilt angle	ϕ	$[\pi/2, \pi]$
Pan angular velocity	$\dot{\psi}$	$[0^\circ/s, 100^\circ/s]$
Tilt angular velocity	$\dot{\phi}$	$[0^\circ/s, 100^\circ/s]$

two-dimensional workspace $\mathcal{W} \subset \mathbb{R}^2$ as depicted in Fig. 1c.Points within this workspace, set in the inertial XY frame, are represented by the two-dimensional vector $\mathbf{x}_T = [x_T, y_T]^T$. The PT camera has a field-of-view (FOV) that is both dynamic and constrained, denoted as $\mathcal{S}(t) \subset W$ at time t. Given an accurate simulation of the camera’s FOV, we can further optimize additional downstream tasks.

Object detection. Object detection, a key computer vision technique, aims to identify and locate objects in images or videos, primarily using convolutional neural networks (CNNs). Leading algorithms such as YOLO (You Only Look Once) [1], SSD (Single Shot MultiBox Detector) [6], and Faster R-CNN (Region-Based Convolutional Neural Network) [7]

enable accurate recognition and classification, crucial for applications like surveillance and autonomous driving.
Object tracking. Object tracking involves continuously locating a moving object over time within a video frame sequence. This technique, crucial for many video-based applications, leverages algorithms to follow the trajectory of an object, even in dynamically changing environments. Depending on the dynamics of the target of interest, common algorithms for object tracking include the Kalman Filter [8], optical flow [9], SORT (Simple Online and Realtime Tracking) [2], DeepSORT (Simple Online and Realtime Tracking with a Deep Association Metric) [10], and ROLO (Recurrent ROLO) [11].
Object classification. In computer vision, object classification is the process of identifying and categorizing objects within an image into groups (or classes) that share certain characteristics. In this application, we employ object classification to distinguish between static and dynamic targets in the inertial frame. This task is non-trivial as the camera's varying pan and tilt angles create a scenario where objects may seem to move within the virtual FOV even if they are static in the inertial frame. The most common techniques for this specific classification task include optical flow, Kalman filtering and kinematic matching.
Camera control. Camera control involves dynamically adjusting camera parameters such as the pan and tilt angles to obtain the desired FOV. This process is crucial for tasks such as object tracking, ensuring the camera maintains the best possible view of targets or areas of interest. Depending on the application, there are several PT control algorithms, including PID (Proportional-Integral-Derivative) control, Kalman filtering, Model Predictive Control (MPC), visual servoing, and other automated pan and tilt control algorithms.

In this project, we model the field of view (FOV) of a Pan-Tilt (PT) camera and implement several key algorithms for object detection, tracking, classification, and camera control. The following sections will introduce the methods used and discuss the results obtained from our simulations, highlighting the accuracy and effectiveness of our approaches.

Methods

In this section, we discuss the specific algorithms and techniques used for modeling the Pan-Tilt (PT) camera's field of view, including YOLOv2 for object detection, the SORT algorithm for tracking, and kinematic matching for target classification. Additionally, we describe our approach to camera control, aimed at optimizing target tracking and maintaining continuous observation within the camera's FOV. The implementation code can be found at `https://github.com/moprescu/MAE5810-Final`.

Pan-Tilt Camera FOV Modeling and Kinematics

We first give a brief description of the techniques used for the camera's FOV modeling. For a more detailed exposition, we refer the reader to part I of this technical report [12].

The PT camera FOV modeling has two central components: 1) establishing a mapping between the inertial frame and the virtual image plane, and 2) incorporating camera kinematics. The relationship between the inertial frame and the virtual image plane (Fig. 1c) is governed by the pan and tilt angles of the camera, given by the Euler angles yaw (ψ) and

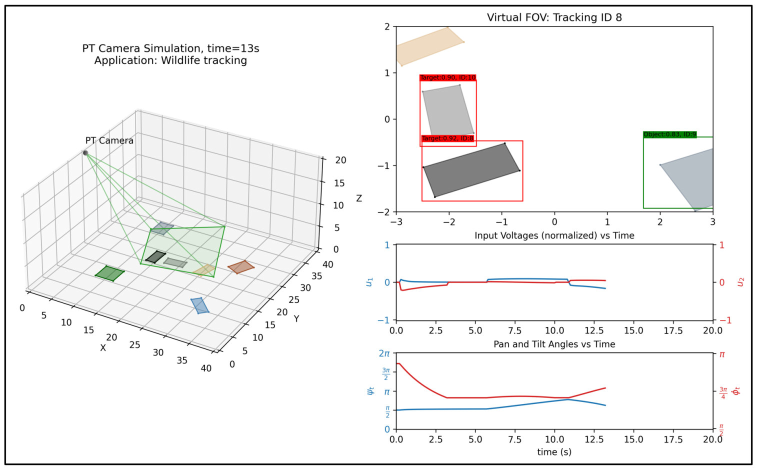


Figure 2: Snapshot of one frame in the wildlife tracking simulation. The left hand side of the figure displays the camera FOV in the workspace $\mathcal{W}$, whereas the right hand side depicts the virtual sensor as well as the camera voltage inputs over time.

roll (ϕ), respectively. To transition from the inertial frame to the camera-fixed frame, we execute two consecutive right-hand rotations which allows us to convert any vector $\mathbf{x}_T \in \mathcal{W}$ into the corresponding camera-fixed frame vector $\mathbf{q}_T$:

$$\mathbf{q}_T = \mathbf{R}_\phi \mathbf{R}_\psi ([\mathbf{x}_T^T \;\; 0]^T - \mathbf{x}_C) \tag{1}$$

Here, $\mathbf{x}_C$ denotes the 3D coordinates of the camera in the inertial frame, while $\mathbf{R}_\phi$ and $\mathbf{R}_\psi$ represent the rotation matrices for the tilt and pan angles, respectively:

$$\mathbf{R}_\phi \triangleq \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{pmatrix}, \quad \mathbf{R}_\psi \triangleq \begin{pmatrix} \cos\psi & \sin\psi & 0 \\ -\sin\psi & \cos\psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{2}$$

The projection $\mathbf{p}_T \in \mathbb{R}^2$ of $\mathbf{q}_T$ onto the virtual image (with focal length $f = \lambda$) is given by:

$$\mathbf{p}_T = \lambda \begin{bmatrix} \frac{(\mathbf{q}_T)_x}{(\mathbf{q}_T)_z} & \frac{(\mathbf{q}_T)_y}{(\mathbf{q}_T)_z} \end{bmatrix} \tag{3}$$

where $(\mathbf{q}_T)_x, (\mathbf{q}_T)_y, (\mathbf{q}_T)_z$ represent the x, y, z coordinates of $\mathbf{q}_T$, respectively. While we forego details here, we note that we can also perform the inverse transformation, as elaborated in part I of this technical report [12].

The geometry of the FOV $\mathcal{S}(t)$ varies with the camera state, yet its projection onto the virtual image plane always forms a rectangle, matching the image sensor's dimensions. In Fig. 2, we depict the true FOV, as well as the virtual sensor image in a snapshot of the wildlife tracking simulation. Denoting the sensor's width and height as a and b, respectively, a target point in $\mathcal{W}$ falls within the camera's FOV if and only if $\mathbf{p}_T$ lies within these sensor dimensions, i.e.:

$$\mathbf{x}_T(t) \in \mathcal{S}(t) \Leftrightarrow \mathbf{p}_T \in [-a/2, a/2] \times [-b/2, b/2] \tag{4}$$

We now give the camera measurement equations for a moving target. Consider a target with a velocity $\mathbf{v}_T = \frac{d\mathbf{x}_T}{dt} \triangleq \dot{\mathbf{x}}_T$ in $\mathcal{W}$. The velocity of the target in the virtual plane is given by:

$$\dot{\mathbf{p}}_T = [\dot{p}_x \;\; \dot{p}_y] = \mathbf{H} \begin{bmatrix} \mathbf{R}_\phi \mathbf{R}_\psi & \mathbf{0} \\ \mathbf{0} & -\mathbf{R}_\phi \end{bmatrix} \begin{bmatrix} \dot{\mathbf{x}}_T^T & 0 & \dot{\phi} & 0 & \dot{\psi} \end{bmatrix}^T \tag{5}$$

where $\mathbf{H}$ is the image Jacobian matrix given by

$$\mathbf{H} \triangleq \begin{bmatrix} -\frac{\lambda}{q_z} & 0 & \frac{p_x}{q_z} & \frac{p_x p_y}{\lambda} & -\frac{\lambda^2 + p_x^2}{\lambda} & p_y \\ 0 & -\frac{\lambda}{q_z} & \frac{p_y}{q_z} & \frac{\lambda^2 + p_x^2}{\lambda} & -\frac{p_x p_y}{\lambda} & -p_x \end{bmatrix}$$

Thus, the velocity in the virtual plane is determined by the target's velocity in $\mathcal{W}$ and the PT camera's kinematics. We next describe the camera's kinematics.

The kinematic equations for the PT camera are derived from motion constraints on the motors and lens, influencing the FOV $\mathcal{S}(t)$. The camera's dynamic state is represented by $\mathbf{s} = [\psi; \phi; \dot{\psi}; \dot{\phi}]^T$, with inputs $\mathbf{u} = [u_1; u_2]^T$. The inputs u_1, u_2, normalized to $[0, 1]$, are voltages controlling the pan and tilt angles independently. The kinematic equations are thus given by $\mathbf{s}(k+1) = \mathbf{A}\mathbf{s}(k) + \mathbf{B}\mathbf{u}(k)$, where:

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & \Delta_t & 0 \\ 0 & 1 & 0 & \Delta_t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ b_1 & 0 \\ 0 & b_2 \end{bmatrix} \tag{6}$$

In the simulation, Δ_t denotes the time step, and $b_1, b_2 = 100°/Vs$ are constants representing motor parameters that limit the maximum pan and tilt angular velocities, detailed in Table 1. Using this information, the complete set of camera constraints is as follows:

$$\begin{cases} \mathbf{s} \geq [0 \;\; \pi/2 \;\; -\dot{\psi}_{\max} \;\; -\dot{\phi}_{\max}]^T \\ \mathbf{s} \leq [2\pi \;\; \pi \;\; \dot{\psi}_{\max} \;\; \dot{\phi}_{\max}]^T \\ \mathbf{u} \leq [1 \;\; 1]^T \end{cases} \tag{7}$$

Object Detection

Next, we discuss the object detection task. In this project, we employed the YOLOv2[1] [1] (You Only Look Once, Version 2) model for object detection. YOLOv2 is a single-stage, real-time object detection model for classification and bounding box predictions. At its core, YOLOv2 relies on a 19-layer convolutional neural network to produce both bounding box and class predictions. Unlike traditional models that process an image in multiple stages, YOLOv2 applies a single neural network to the whole image, dividing it into regions and predicting bounding boxes and probabilities for each region simultaneously. This single-step approach allows YOLOv2 to detect objects rapidly and accurately. For our purpose, YOLOv2's capability to handle diverse environmental conditions makes it an ideal choice. Its

[1]Despite the availability of more advanced YOLO models beyond YOLOv2, we determined that YOLOv2 strikes an optimal balance for our project, offering the necessary accuracy for our task while maintaining computational efficiency.

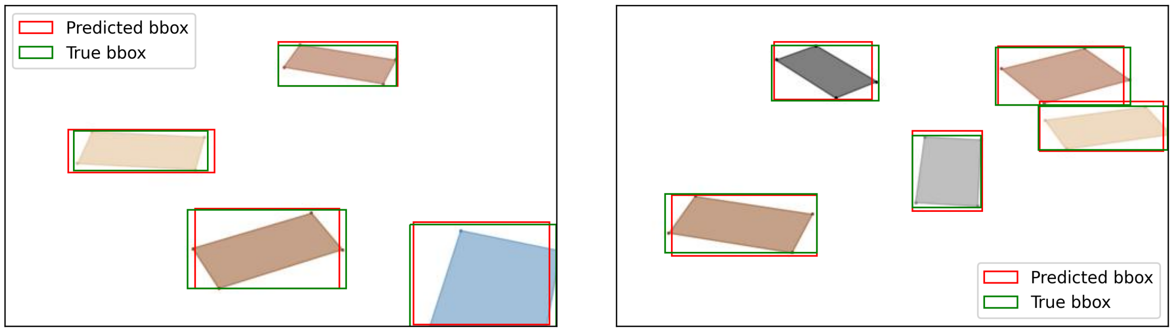


Figure 3: Illustration of YOLOv2 performance on two test FOV images. The green boxes depict the bounding boxes generated with the training data, and the red boxes show the bounding boxes returned by the trained YOLOv2 instance.

robust detection capabilities across various lighting and terrains are essential for effectively monitoring wildlife like Snow Leopards which is known to blend well into their background. Additionally, its real-time processing enables prompt detection of moving animals, facilitating timely and accurate data collection for conservation.

Data generation for YOLOv2. To adapt YOLOv2 for detecting our specific objects of interest, we must train it using data generated from our camera's FOV simulations. For this purpose, we designed three distinct environments in our simulations, each with varying appearances of static objects and behaviors of moving targets. Additionally, we incorporated three types of camera motions: two variations of pan-only motions with different constant tilt angles, and one tilt-only motion with a fixed pan angle. For each combination of environment and camera motion, we sampled 100 frames, resulting in a total of 900 frames. After removing frames without objects, we were left with 740 frames. The training dataset includes simulations from two environments (approximately 490 frames), while the remaining simulations from the third environment (around 250 frames) are used for testing. For each frame, we generated bounding box coordinates corresponding to the objects in the simulations. These coordinates, along with the raw images, form the training and testing datasets for YOLOv2.

YOLOv2 training. In order to decrease the computational overhead of netweork training, we started with a set of YOLOv2 pretrained weights. This process is known as transfer learning, where a model developed for one task is reused as the starting point for a model on a second task. This approach is known to significantly reducing the computational cost and data requirements for training, and often leads to improved performance on the new task. We trained the YOLOv2 on our dataset using Google Colab [13] due to the availability of a high performance GPU, namely the T4 GPU. We trained the network for 3,000 iterations which amounted to around 3 hours of training. The final error rate was 0.0923 mAP (mean average precision) with an average region IoU (Intersection over Union) of 0.85. In **??**, we depict the outputs of YOLOv2 on two test FOV frames. We note that the algorithm returns very accurate predictions, including for edge cases such as partially observed objects.

In conclusion, our use of transfer learning with YOLOv2, supported by Google Colab's GPU, enabled efficient training and effective object detection in our wildlife monitoring simulations.

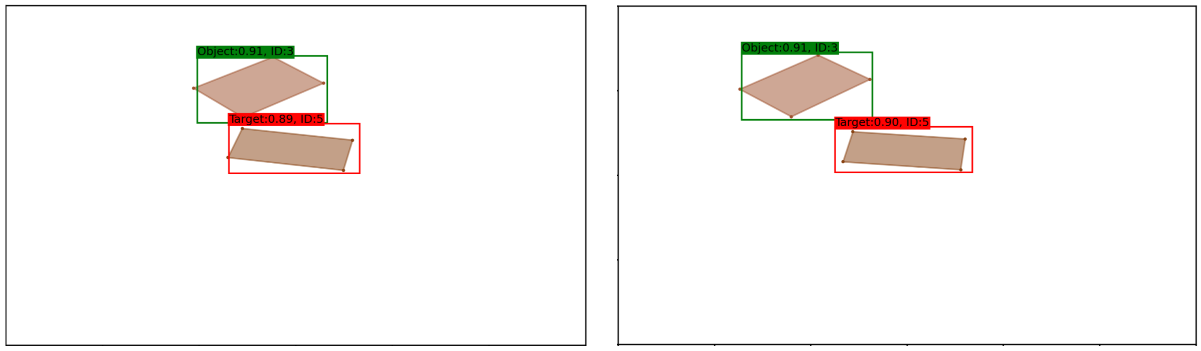


Figure 4: Illustration of SORT's performance across two FOV frames captured at an interval of five time steps of the wildlife monitoring simulation. The bounding boxes are generated by YOLOv2 and the object IDs are assigned by SORT. Green boxes depict static objects whereas red boxes depict a moving target.

Object Tracking

For the tracking task, we implemented the SORT (Simple Online and Realtime Tracking) algorithm, chosen for its optimal balance between simplicity and performance efficiency. SORT is specifically tailored for object tracking in video sequences and is designed to prioritize real-time processing, emphasizing rapid and efficient tracking capabilities. The speed and simplicity follows from the fact that SORT only relies on bounding box information, deliberately omitting the use of appearance features that typically necessitate a trained neural network. The core of the algorithm is a Kalman filter [8] with a linear constant velocity model between frames. The Kalman filter model assumes that the true state at time k is derived from the state at time $(k-1)$, following the following relationship:

$$\mathbf{x}_k = \mathbf{F}_k \mathbf{x}_{k-1} + \mathbf{B}_k \mathbf{u}_k + \mathbf{w}_k \tag{8}$$

where $\mathbf{F}_k$ is the state transition model (corresponding to a linear velocity model in SORT), $\mathbf{B}_k$ is the control-input model applied to the control vector $\mathbf{u}_k$, and $\mathbf{w}_k$ is the process noise.

The SORT algorithm then uses the predicted next frame positions in conjunction with the actual bounding boxes from the next frame to accurately associate predictions with specific objects. This association process employs the Hungarian algorithm [14] with threshold based on the IoU metric between the predicted and actual bounding boxes to ensure precise matching. In Fig. 4, we depict the predictions of the SORT algorithm on two frames captured at an interval of five time steps of the same simulation. We note that the SORT algorithm accurately assigns IDs to the targets in the two frames. Thus, the integration of the SORT algorithm in our system demonstrates effective and efficient object tracking across FOV frames, proving essential for accurate wildlife monitoring and tracking.

Object Classification

To construct an accurate simulation of wildlife tracking and monitoring, we wish to distinguish static objects from moving targets within the inertial frame using only FOV images. This task is complex, as the kinematics of objects in the FOV result from a composition of the camera's varying pan and tilt angles, with the objects' motion in the inertial frame.

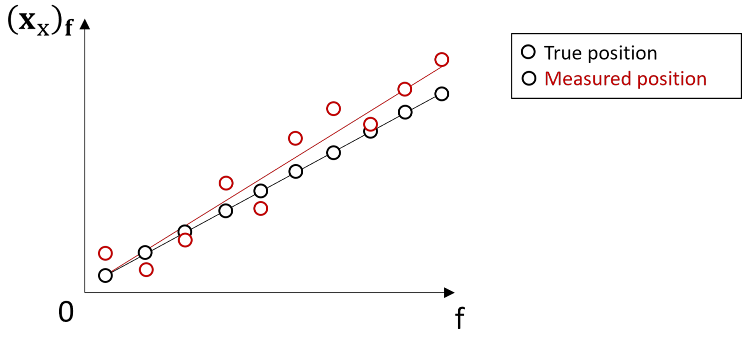


Figure 5: Illustration of the kinematic matching approach. $\mathbf{x}_x$ represents the x-coordinate of the object in the inertial frame, whereas f represents the frame number.

The first step towards accomplishing this task is to estimate the velocities of objects in the inertial frame for which we utilize kinematic matching. Kinematic matching algorithms treat target tracking as a parameter estimation problem, initially hypothesizing a parametric model of the target's motion and subsequently identifying parameter values that most accurately align with observed data. Commonly in various applications (including ours), the optimal fit is determined using a batch ordinary least squares (OLS) estimator [5]. We now provide the calculations required for kinematic matching. Suppose we had a set of measurements of the object position in the inertial frame:

$$\mathbf{z}^f = \begin{bmatrix} \mathbf{z}_1^T \\ \cdots \\ \mathbf{z}_f^T \end{bmatrix} \in \mathbb{R}^{f\times 3} \tag{9}$$

where f is the frame number indicator and $\mathbf{z}_i$ is a measurement of the 3D coordinates of the object at frame index i. We posit the following parametric model for the inertial velocity at time f, $\dot{\mathbf{x}}_T(f)$:

$$\mathbf{z}^f = \mathbf{H}^f \dot{\mathbf{x}}_T(f) + \mathbf{n}^f \tag{10}$$

where $\mathbf{H}^f$ is the stacked measurement matrix and $\mathbf{n}^f$ is the measurement noise vector which is assumed to be Gaussian and homogeneous. If we further assume that the velocity of the target object is constant, $\mathbf{H}^f$ is given by:

$$\mathbf{H}^f = \begin{bmatrix} f - f_0 \\ f - f_0 \\ \cdots \\ f - f_0 \end{bmatrix} \in \mathbb{R}^{f\times 1} \tag{11}$$

where f_0 is the frame index of the first measurement. Assuming that the samples in the batch are equally weighted, the OLS solution (ordinary least squares) is given by:

$$\hat{\dot{\mathbf{x}}}_T(f) = [(\mathbf{H}^f)^T \mathbf{H}^f]^{-1} (\mathbf{H}^f)^T \mathbf{z}^f \tag{12}$$

Thus, assuming access to $\mathbf{z}^f$, kinematic matching will return an estimate for the velocity in the inertial frame of a target object. We illustrate this process in Fig. 5. However, we do not directly observe $\mathbf{z}^f$. Instead, we observe a proxy $\mathbf{p}^f$ of object positions on the virtual FOV. To obtain the inertial frame position estimates, we apply the virtual-to-inertial frame transformation given in [12]:

$$\mathbf{z}_i^T = \left[(\mathbf{x}_C)_x - \frac{\left(\mathbf{R}_\psi^T \mathbf{R}_\phi^T \begin{bmatrix} \frac{\mathbf{p}_i^T}{\lambda} & 1 \end{bmatrix}\right)_x \cdot (\mathbf{x}_C)_z}{\left(\mathbf{R}_\psi^T \mathbf{R}_\phi^T \begin{bmatrix} \frac{\mathbf{p}_i^T}{\lambda} & 1 \end{bmatrix}\right)_z} \quad (\mathbf{x}_C)_y - \frac{\left(\mathbf{R}_\psi^T \mathbf{R}_\phi^T \begin{bmatrix} \frac{\mathbf{p}_i^T}{\lambda} & 1 \end{bmatrix}\right)_y \cdot (\mathbf{x}_C)_z}{\left(\mathbf{R}_\psi^T \mathbf{R}_\phi^T \begin{bmatrix} \frac{\mathbf{p}_i^T}{\lambda} & 1 \end{bmatrix}\right)_z} \quad 0 \right]^T \tag{13}$$

In our simulations, we chose the $\mathbf{p}_i$'s to correspond to the lower left corner of the bounding box in order to avoid erroneous early measurements when the object is only partially observed.

The final step of our method is a simple classification algorithm that categorizes an object as a moving target if and only if the estimated inertial velocity $\widehat{\dot{\mathbf{x}}}_T(f)$ has an euclidean norm above a threshold v_{th}. For this application, we chose a threshold of $v_{\text{th}} = 0.02$ units/frame where the "units" are the simulation units in the inertial frame. In other words, we classify an object as a moving target if and only if $\|\widehat{\dot{\mathbf{x}}}_T(f)\| \geq 0.02$ units/frame. With this threshold, we achieve an classification accuracy of 0.98 (98%) averaged over all objects and 100 frames of a simulation. In our simulation (see example in Fig. 4), we illustrate the result of the classification algorithm by coloring the bounding box with **green** if the object is classified as "object" (i.e. static in the inertial) and **red** if the object is classified as "target" (i.e. moving in the inertial frame).

To conclude this section, kinematic matching and thresholding effectively classify static and moving objects in wildlife monitoring, significantly enhancing the precision of our tracking system.

Camera Control and Target Pursuit

In this section, we describe a simple strategy used to dynamically adjust our camera system for optimal target tracking. The goal of this task is to maintain continuous observation on a chosen moving target within the camera's field of view. In other words, we wish to create a mapping of the target's inertial velocity to camera controls, i.e. $\dot{\mathbf{x}}_T \mapsto [\dot{\psi}, \dot{\phi}]$, such that the target remains within the FOV of the PT camera. A simple algorithm for producing this mapping is to require the projection of the virtual FOV's center onto the inertial frame to kinematically match the target object. We can achieve this by noting the following system of equations holds for any $\mathbf{q}_T, \dot{\mathbf{q}}_T$ camera frame states of a target in the inertial frame (with $\dot{\mathbf{x}}_T$ inertial velocity):

$$\begin{bmatrix} (\dot{\mathbf{x}}_T)_x \\ (\dot{\mathbf{x}}_T)_y \\ 0 \end{bmatrix} = \mathbf{R}_\psi^T \mathbf{R}_\phi^T \dot{\mathbf{q}}_T + (\dot{\mathbf{R}}_\psi^T \mathbf{R}_\phi^T \dot{\psi} + \mathbf{R}_\psi^T \dot{\mathbf{R}}_\phi^T \dot{\phi}) \mathbf{q}_T \tag{14}$$

where $\dot{\mathbf{R}}_\psi, \dot{\mathbf{R}}_\phi$ are given by:

$$\dot{\mathbf{R}}_\phi \triangleq \begin{pmatrix} 0 & 0 & 0 \\ 0 & -\sin\phi & \cos\phi \\ 0 & -\cos\phi & -\sin\phi \end{pmatrix}, \quad \mathbf{R}_\psi \triangleq \begin{pmatrix} -\sin\psi & \cos\psi & 0 \\ -\cos\psi & -\sin\psi & 0 \\ 0 & 0 & 0 \end{pmatrix} \tag{15}$$

Table 2: Simulation parameters

Description	Variable	Value
$t_{\max}$	Maximum simulation time	$20s$
fps	Frames per second	10
$(x_{\max}, y_{\max})$	Workspace axis limit	$(40, 40)$
$\mathbf{x}_c$	Camera position	$[0 \;\; 20 \;\; 20]^T$
λ	Camera focal length	10
(a, b)	Sensor width and height	$(6, 4)$
(T_w, T_h)	Moving target width and height (rectangle)	$(4, 3)$
(O_w, O_h)	Static object width and height (rectangle)	variable

This equation was obtained by taking the derivative w.r.t. time of both sides of Eq. (1) and rearranging the terms. Taking $\mathbf{p}_T = [0 \quad 0]^T$ and $\dot{\mathbf{p}}_T = [0 \quad 0]^T$ for the FOV center, Eq. (14) has three unknowns, $(\dot{\mathbf{q}}_T)_z, \dot{\psi}, \dot{\phi}$, that we can solve for. Thus, this process enables to create the control desired mapping. The final step of the algorithm is to incorporate the camera constraints in Table 1, as well as the constraint that the FOV cannot be more than 50% outside of the monitored workspace. The last constraint, although seemingly arbitrary, ensures that we maintain focus on the monitored area.

As demonstrated in the simulation videos hosted at github.com/moprescu/MAE5810-Final, our camera control and target pursuit strategies effectively ensure continuous, precise tracking in wildlife monitoring. Thus, these techniques have an essential role in efficiently gathering longitudinal behavioral data from wildlife targets.

Results

We summarize the fixed simulation parameters, as well as camera and object/target characteristics in Table 2. Our simulations run for $t_{\max} = 20s$ with a frame-per-second count of 10. The simulation display includes four figures (see Fig. 2 for an example): 1) the 3D inertial frame that includes the camera and the workspace with the true FOV projection, targets, and environment features, 2) the virtual sensor image with objects, targets, and corresponding bounding boxes marked with the detection accuracy, object ID, and classification result, 3) a plot of the input voltages over time, and 4) a plot of the pan and tilt angles over time. These figures are updated with each frame.

We developed two distinct simulation scenarios: first, a wildlife monitoring simulation without intelligent camera control, aimed at maximizing coverage of the workspace within a set timeframe, and second, a wildlife tracking scenario incorporating intelligent camera control, focused on continuously tracking a chosen target until it exits the workspace, and then repeating this process for the duration of the simulation. In both scenarios, the environment includes four static objects, each varying in color, shape, position, and orientation, alongside six moving targets that enter in two batches, one at $t = 0s$ and the other at $t = 10s$. These moving targets share the same shape but differ in color and velocity within the inertial frame.

In Fig. 6 below, we display two frames $1s$ apart from the tracking simulation:

The two frames effectively demonstrate the efficacy of the methodologies detailed in this

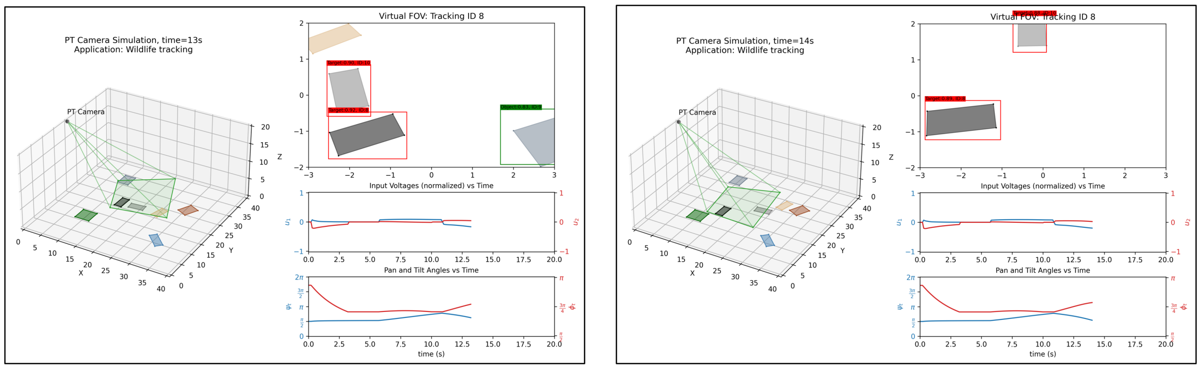


Figure 6: Two snapshots of frames from the tracking simulation. The frames are obtained approximately 1s apart.

report. Specifically, they highlight the system's precision in object identification within the camera's field of view, as evidenced by the accurate bounding boxes, consistent tracking with stable object IDs across frames, reliable classification depicted through the color-coded boxes indicating static or moving objects in the inertial frame, and effiecient camera control, showcased by the camera's pan and tilt adjustments to keep the identified target (ID 8) within the virtual FOV.

Conclusion

This technical report successfully demonstrates the integration and effectiveness of advanced computer vision techniques in wildlife monitoring and tracking applications. By employing algorithms like YOLOv2 for object detection, SORT for tracking, and kinematic matching for classification, together with dynamic camera control strategies, we have developed a robust system capable of accurately monitoring wildlife. The simulations in varied scenarios underline the system's adaptability and precision in different environmental conditions. Thus, this work offers promising tools for conservation efforts and biodiversity studies. The methodologies and findings from this project have potential applications in broader domains, paving the way for more sophisticated and efficient monitoring systems in the future.

References

[1] Joseph Redmon and Ali Farhadi. Yolo9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[2] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple online and realtime tracking. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 3464–3468, 2016.

[3] Snow leopard conservancy. `https://snowleopardconservancy.org/`. Accessed: 2023-10-12.

[4] T. McCarthy, D. Mallon, R. Jackson, P. Zahler, and K. McCarthy. Panthera uncia. *IUCN Red List of Threatened Species*, 2017.

[5] Silvia Ferrari and Thomas A Wettergren. *Information-driven planning and control.* MIT Press, 2021.

[6] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, pages 21–37. Springer, 2016.

[7] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

[8] JR Fisher. Optimal nonlinear filtering. In *Advances in Control Systems*, volume 5, pages 197–300. Elsevier, 1967.

[9] Berthold KP Horn and Brian G Schunck. Determining optical flow. *Artificial intelligence*, 17(1-3):185–203, 1981.

[10] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. Simple online and realtime tracking with a deep association metric. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 3645–3649. IEEE, 2017.

[11] Guanghan Ning, Zhi Zhang, Chen Huang, Zhihai He, Xiaobo Ren, and Haohong Wang. Spatially supervised recurrent convolutional neural networks for visual object tracking. *arXiv preprint arXiv:1607.05781*, 2016.

[12] Miruna Oprescu. Pan-Tilt Camera Field-of-View Modeling. Technical report, Cornell University, Cornell Tech, Oct 2023.

[13] Ekaba Bisong and Ekaba Bisong. Google colaboratory. *Building machine learning and deep learning models on google cloud platform: a comprehensive guide for beginners*, pages 59–64, 2019.

[14] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.