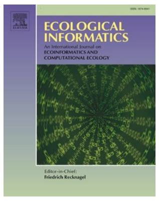
Contents lists available at ScienceDirect

Ecological Informatics

journal homepage: www.elsevier.com/locate/ecolinf



Motion vectors and deep neural networks for video camera traps

Miklas Riechmann[a,b], Ross Gardiner[a,b], Kai Waddington[a,c], Ryan Rueger[a,d], Frederic Fol Leymarie[a,e], Stefan Rueger[a,c,*]

[a] DynAIkon Ltd., London, UK
[b] University of Glasgow, James Watt School of Engineering, UK
[c] The Open University, Knowledge Media Institute, UK
[d] ETH Zürich, Department of Mathematics, Switzerland
[e] Goldsmiths, University of London, Department of Computing, UK

ARTICLE INFO

Keywords:
AI
Computer vision
Machine learning
Motion vectors
Video camera trap
Video pipeline

ABSTRACT

Commercial camera traps are usually triggered by a Passive Infra-Red (PIR) motion sensor necessitating a delay between triggering and the image being captured. This often seriously limits the ability to record images of small and fast moving animals. It also results in many "empty" images, e.g., owing to moving foliage against a background of different temperature. In this paper we detail a new triggering mechanism based solely on the camera sensor. This is intended for use by citizen scientists and for deployment on an affordable, compact, low-power Raspberry Pi computer (RPi). Our system introduces a video frame filtering pipeline consisting of movement and image-based processing. This makes use of Machine Learning (ML) feasible on a live camera stream on an RPi. We describe our free and open-source software implementation of the system; introduce a suitable ecology efficiency measure that mediates between specificity and recall; provide ground-truth for a video clip collection from camera traps; and evaluate the effectiveness of our system thoroughly. Overall, our video camera trap turns out to be robust and effective.

1. Introduction

In the world of camera trapping for biodiversity monitoring one of the greatest problems is the large quantity of data that is generated. Many of the captured images do not even contain any animals, and so significant time needs to be spent on simply removing empty images. Large advances have been made in this regard, especially with the introduction of Machine Learning (ML) into this field (Tabak et al., 2019; Wei et al., 2020; Xi et al., 2021). The standard approach is one of retroactively removing empty images. We propose a solution where instead our camera trap system does not capture empty images in the first place. Some advantages of our approach include reducing: (i) bandwidth requirements if live data is sent; (ii) storage requirements when saving data to disk; and (iii) assessment time at later stages of analysis. We provide the open-source code for the camera trap, documentation with a setup guide, video evaluation corpus, and our evaluation library at https://dynaikon.com/trap. The DIY setup consists of downloading our software to an RPi with camera; the full setup takes under an hour.

1.1. Aims and objectives

The main aim is to develop a camera trap that captures video clips or images when an animal has entered the camera's Field of View (FoV) without relying on a Passive Infra-Red (PIR) sensor. We achieve this by introducing Artificial Intelligence (AI) to the camera trap. The issues with PIR sensors are caused by the FoV of the sensor and camera not perfectly overlapping, delays in responding to a motion signal, and physical limitations of PIR sensors relating to animal speed/size and background movement. The desired outcomes are to:

1. significantly reduce the number of empty images captured;
2. facilitate animal detection and identification;
3. provide an affordable system for researchers and citizen scientists.

Our DynAIkonTrap project has two strands: hardware and software.

* Corresponding author at: DynAIkon Ltd., London, UK.
E-mail addresses: mhr@dynaikon.com (M. Riechmann), ross.gardiner@dynaikon.com (R. Gardiner), kai@dynaikon.com (K. Waddington), rrueger@dynaikon.com (R. Rueger), ffl@dynaikon.com (F.F. Leymarie), smr@dynaikon.com (S. Rueger).

https://doi.org/10.1016/j.ecoinf.2022.101657
Received 9 September 2021; Received in revised form 14 January 2022; Accepted 25 April 2022
Available online 13 May 2022

The hardware strand is concerned with designing an inexpensive platform with an RPi computer at the centre that has wireless connectivity to enable automated upload of observations to user-chosen platforms such as the European Open Science Cloud (EOSC) or national citzen observatories such as iSpot,[1] and providing a power supply that harvests energy (Proppe et al., 2020) rather than relying on mains electricity (a.k.a. utility power). The platform is also to be capable of taking environmental readings to give context to the visual observations.

Our open-source software is focused on determining the best camera-based triggering techniques for this application by introducing AI into the camera trap.

1.2. Scope and structure

The scope of this paper encompasses the methods under the software strand of this project, as this is where novel ideas are explored. Our approach lends itself to producing video clips and/or still images of animal observations. We therefore use the words image and (video) frame interchangeably throughout this article.

Section 2 first provides an overview of recent camera trap systems. Based on that background, we then give the main design goals for our proposed smart camera trap system (Section 2.2). Section 3 presents the architecture of our system, while Sections 4 and 5 detail our movement detection method through motion vectors and our animal detection filter using AI. The overall system is evaluated in different configurations in Section 6.

2. Towards smart camera traps

2.1. Background

Traditional camera traps use PIR sensors as a motion-based trigger; these respond to changes in infrared radiation due to a body of heat moving in front of a background with a different temperature. Findlay et al., 2020 suggest these sensors can fail to trigger due to the distance of an animal from the sensor, or even short-term changes in the animal's skin temperature due to wetness. They also mention the well-known negative impact of animal speed on detection-probability (Glen et al., 2013; McIntyre et al., 2020). Cameras with a lower trigger delay somewhat reduce this effect (Robley et al., 2010). Driessen et al. (2017) compared four camera traps using triggers and visits (by the same animal) as metrics, following Meek et al.'s (2014a) recommendations. They concluded that detection zones have a higher influence on performance than trigger speed.

In the past, using a white camera flash has been seen to interfere with animals (Glen et al., 2013), often startling them. Some recent research by Taggart et al. (2020) did not find this to be the case for the specific case of feral cats when using infrared camera flashes. It is, however, generally thought that camera traps can influence animal behaviour (Meek et al., 2016), with some species being attracted and others repelled by their presence. This is not just limited to visual effects of the traps: there are audible aspects owing to electronic components in the traps (Meek et al., 2014b). One possible cause for noise is the piezoelectric effect in multilayer ceramic capacitors when mounted on a circuit board (Ko et al., 2014).

Camera traps used to generate the well known *Snapshot Serengeti* (Swanson et al., 2015) dataset were triggered around 1.2 million times, with 76% of these being empty. It was reported that poor quality nighttime images prompted a switch from DLC Covert II cameras (with infrared flash), to Scoutguard SG565 cameras (with white flashes). An evaluation of six camera trap models by Weingarth et al. (2013) covered a range of tests including image quality, trigger speed, ease of use, and more. They describe a robust test setup carried out under laboratory conditions. Results indicate the achievable camera coverage is often lower than the range specified by the manufacturer. All six tested cameras had an actual range of either 7 m or 8 m, despite manufacturer claims of up to 18 m for one model. The underlying point behind their conclusions is that it is difficult to make a general recommendation for a particular trap. The decision of which camera to choose depends on various factors, such as image quality, the necessary range, or the available shelter from direct sunlight, as well as budget.

Many of the issues identified thus far have been shown to lead to incorrect estimates of populations and visitation frequency. Jumeau et al. (2017) found that 43% of mammals were missed, whilst Urbanek et al. (2019) found traps missed, depending on species, between 14% (bears) to 92% (squirrels) of triggers. This has been attributed to the use of PIR sensors as their performance is largely dependent on ambient temperature, wind, and size of the animals.

Other triggering mechanisms have been tried as alternatives to PIR. Microwave-based sensors do not perform as well as PIR (Glen et al., 2013). Light-gates have been used effectively for small animals by Hobbs and Brehme (2017), although their approach cannot easily be scaled up for larger animals. Manufacturers of commercially available camera traps therefore usually still opt for triggering based on PIR sensors.

Another more recent trend in designing camera traps is to use flexible low-power, low-cost, fully programmable platforms. The most versatile hardware allows connecting different camera types in terms of focal length or infrared sensitivity. It is also very easy to include code for automated upload to citizen observatories, cloud storage or similar for a seamless workflow. The Raspberry Pi (RPi) series is the current most popular inexpensive and flexible set of hardware platforms, with an established wide reach in Education and Science in general, with many applications in Biology in particular (Jolles, 2021).

The WiseEye system is based on a RPi 2B with compatible peripherals (Nazir et al., 2017). It introduces the concept of "confirmatory sensing", in which the PIR triggering is confirmed through two other modalities (radar and pixel change using background subtraction) to reduce the occurrence of false positives images. The code base is open source.

PICT (plant–insect interactions camera trap) is a do-it-yourself system based on a RPi Zero designed to continuously film small animal activity at close range (Droissart et al., 2021). Lower energy consumption is improved by separating the recording from the (offline) analysis steps.

Klemens et al. (2021) use an RPi always turned on, with an infrared camera combined with a traditional external motion detection method to avoid latency issues of PIR-based systems. They tested their system only in the specific scenario of detecting flying squirrels at night, with a fixed background that is needed by their motion detection software.

Recently, Artificial Intelligence (AI) has been brought to some commercial camera traps, such as the *Trail-Guard* by Resolve, although literature around effectiveness for this is lacking. The use of AI "at the edge" may be able to overcome some of the issues of traditional camera traps, such as a large number of empty images being kept for analysis and later discarded. More specifically, machine learning, especially based on deep learning techniques, is being explored by the research community to reduce as much as possible the amount of useless data being collected. For example, Schindler and Steinhage (2021) use a combination of Mask R-CNN with Flow-Guided Feature Aggregation (Zhu et al., 2017) to optimize instance segmentation of animal species in video clips. Their study is applied to data obtained by camera traps with PIR, capturing at dusk and night, in Bavaria, four classes of animals (deer, boar, fox and hare). A recent review by Petso et al. (2022) further confirms that machine learning is being explored in various ways in camera trap systems for research purposes.

[1] https://www.ispotnature.org/

2.2. Smart camera trap hardware

While it is conceivable that a simple camera permanently records video for later analysis by a powerful computer elsewhere, there are severe practical limitations in terms of storage capacity or bandwidth: a typical video stream of 10 Mbit/s fills one Terabyte of storage in 10 days requiring the storage card of the camera be frequently retrieved. Alternatively, transmitting a 10 Mbit/s data stream to a processing server requires a correspondingly fast internet. Both are unrealistic for deployment of a camera trap at remote places. A far better option is to process the video stream on the camera trap itself, especially if all necessary energy for the computation and recording is harvested from the environment. Hence, no frequent visits to the camera trap site would be necessary.

The move towards a smart camera trap requires a programmable hardware, where the camera trap itself can make decisions when to record to disk and which clips should be kept. Possible hardware ranges from microprocessor boards to full-fledged low-power computers complete with an operating system, a range of programming languages and hardware headers to interface with an external camera lens and other sensors. Modern smartphones can be turned into rudimentary camera traps by installing an appropriate application, such as “Photo-Trap Trail Camera” or “Motion Detector Pro”.

The most versatile hardware for a smart camera trap is a low-power computer that allows connecting different camera types in terms of focal length or infrared sensitivity. This type of hardware also makes it very easy to include code for automated upload to citizen observatories, cloud storage or similar for a seamless workflow. Our choice fell on the Raspberry Pi, which is inexpensive, has a wide community of users and lends itself well to the implementation of a Computer Vision and Deep Neural Network pipeline as detailed in Section 3.

Invariably, camera trap projects also benefit from environmental sensor data that are recorded together with the observations. Our proposed smart camera trap can interact with an optional sensor board that records temperature, humidity, air pressure, brightness, altitude, GPS position, time, and, derived from this, orientation and height of the sun above horizon. The latter is useful if observations are meant to be restricted to certain times (dawn and dusk, night or day). The granularity of the location recording can be set by the user, which can aid in the protection of rare observed species or for privacy concerns, such as when deploying in a garden. The optional sensor board interacts with the host system via a USB connection and has been published as open hardware[2]; it can be built by the user at moderate cost or be ordered for larger citizen science projects from a PCB house. Again, the choice of an RPi makes it easy to interact with such a sensor board.

While software for the RPi can be fairly easily ported to other computing platforms, our current implementation depends on the availability of a reasonably modern multi-process operating system, the Python programming language with thread support and the open source ffmpeg library to process video streams. The only other dependency is the PiCamera library, which provides a Python interface to the RPi camera module.

3. Solution/system architecture

We propose a camera trap that is always turned on, ready to record video, and detect an event in real time. The needed computational resources for live image analysis, both in terms of software and hardware, have recently become available on low-power devices.

Fig. 1 summarises the software design of our system. The majority of work belongs to the Filter class in the diagram, which includes filters to determine whether or not a camera frame contains an animal. The optional data stream of DynAIkon’s sensor board gets merged at source with interesting images to enrich the observations. The code is designed to be very modular, enabling easy changes to particular aspects, for example modifying the Output to save to disk in a different format. Modularity also permits the configuration of different filters for different use cases. This is particularly relevant for the detection of desired, prevalent or studied animal classes. Modularity also allows adaptation to any platform’s computing power, making this a somewhat future-proof design.

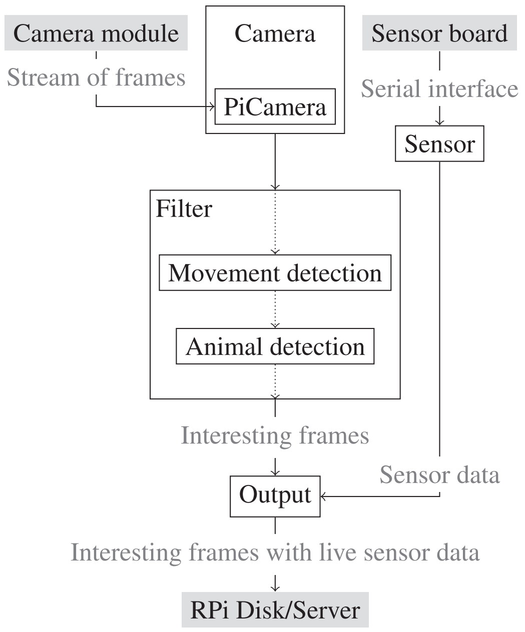


Fig. 1. Data flow through the system; boxes with a black outline indicate software classes/containers, whilst grey boxes represent the hardware depicted in Fig. 2.

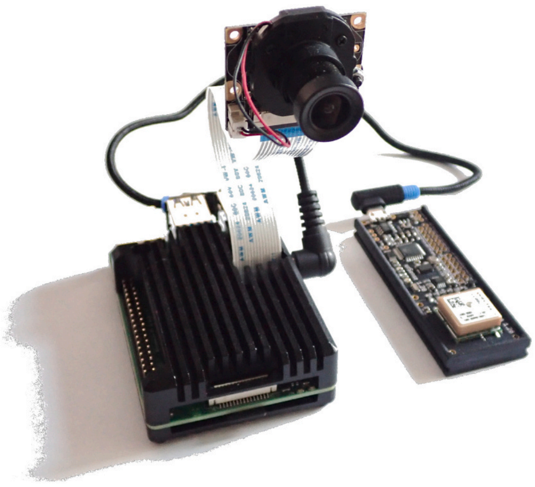


Fig. 2. RPi, camera, and DynAIkon’s sensor board.

The Filter class incorporates a sequence of filters that ultimately lead to a stream of animal images: first a fast movement detector selects potential animal images, which are then analysed by a slower ML-based animal detector. Together with a buffer, this setup allows the real-time analysis of incoming camera frames and reduces the time needed for computation. Both filters run concurrently, which means that near-instant movement detection can always operate, while the slower animal detector’s buffer is allowed to fill up. Sections 4 and 5 detail the inner workings of these two filters.

[2] https://gitlab.dynaikon.com/dynaikontrap/urSense

Within the Camera class, frames are constructed to include both H.264 motion vectors (Section 4) and image data for that frame. The time of capture or *timestamp* is also associated with the frame. This allows sensor data to be matched with the correct camera frame, as the sensor data is captured asynchronously. This Camera class can also be replaced by a module, such as our vid2frames[3] library, which can pass in image and motion frames from pre-recorded videos. This makes our pipeline flexible and usable for the retrospective analysis of videos on non-RPi hardware.

The remainder of the paper frames detection as a filtering task, where the camera trap acts as an animal filter on a stream of images. It is possible that some users will not actually want every image of an animal from a video stream of one observation. In such cases, some additional software would be required to reduce the throughput. The simplest solution would be to send every n^{th} animal image, whilst a more elaborate option is to send the highest-confidence animal image or the one with the lowest motion blur.

4. Movement detection

This section explores existing methods for movement and foreground detection, contrasts this to our novel method, and details our implementation.

4.1. Related work

Movement detection often deploys background subtraction methods, particularly when the camera position is static. The two most prominent background subtraction methods are *MOG2* (mixture of Gaussians, Zivkovic, 2004) and *KNN* (*k* nearest neighbours, Zivkovic and van der Heijden, 2006). A comparison of the two methods by Trnovszký et al. (2017) demonstrated a slightly better accuracy of KNN over MOG2, although MOG2 performed faster. MOG2's faster execution makes it more suited to real-time applications like our designed camera trap.

Many approaches compute pixel-wise differences between successive frames. Small naturally occurring differences between frames will result in noise even in the absence of movement. This noise is often removed by applying morphological opening and closing transformations. One example is the Zilong application (Wei et al., 2020), which uses a combination of detection of colour-change and edges. The edge detection is added to improve the system when dealing with foggy images, while colour-change detection is the central feature. The method is based on a three stage process of applying a mean shift operation to the input images, taking the absolute difference of two consecutive mean-shifted images, and thresholding the result to generate a black and white mask. A simple count of the number of white pixels is used to declare an animal detection if the sum exceeds a threshold value.

Whilst ML approaches exist that could theoretically be used for this task, these models are currently too slow to be used in practice: the purpose of having a multi-filter pipeline is to reduce the set of images that need to be inspected by a ML model.

4.2. Detection methods

Video encoding standards such as H.264 commonly reduce temporal redundancy of the video stream by using motion vectors that encode how pixel patches, of sizes between 4×4 to 16×16 px, move from one frame to the next, thus reducing the need to send full frames all the time. These motion vectors can be extracted from the video stream without fully decoding it, are computed in hardware and therefore can be utilised without further overhead. To the best of our knowledge, utilising these motion vectors in this way has not been integrated into camera trap products until now, nor discussed in the literature. Section 4.2.3 details how we set this up for our camera trap. First, however, we report on the simpler "difference of images" and on background subtraction.

4.2.1. Difference between images

Each frame is consecutively read in and, pixel for pixel, the absolute brightness difference between them is recorded (in contrast to Zilong which considers colour differences). A binary threshold is applied to the result, generating a black and white mask, and the number of white pixels counted. If the sum exceeds a second threshold, an animal detection is declared, otherwise the image is declared empty. This works when there is no change in background pixels and only the foreground (an animal) is in motion. It is worth noting that a moving animal will likely only cause a difference around its outline if the body of the animal has a consistent colour or homogeneous texture. The method quickly breaks down when there are slight variations in the background from one frame to the next, such as due to leaves in the wind. We note that the difference of two images does not express the speed at which a change happened, nor does it express a direction of movement. Both of these are detectable using our motion vector method discussed in Section 4.2.3.

4.2.2. Background subtraction

We explored MOG2 as a representative algorithm of the background subtraction class. The subtractor takes an image and returns a mask image in black and white, the latter indicating foreground. If this mask is applied to the current image, the foreground remains, whilst the background is masked off. As further images are passed in, the subtractor learns the background, meaning the system is able to adjust to changes in the background over time. The generated mask can appear noisy due to large differences of particular pixels in two consecutive images. This will occur as each frame can contain different noise. Applying consecutive opening and closing morphological operations helps in lessening this effect, as shown in Fig. 3.

After these operations, one is left with a mask indicating the potential animal location in the image. The next step is to determine whether there is a sufficient number of white pixels in the image. This issue can be complicated by dynamic backgrounds and large changes in lighting. These can lead to erroneous foreground detections in various patches throughout the mask. We initially tried to overcome this through the use of contour generation functions, whereby a list of regions in the black and white mask can be found. Each region's area is determined, with an animal detection being declared once a region of sufficient area has been found. If no such regions are found, the image is declared as empty.

A simpler approach is to count the number of white pixels present in the mask. The image is then declared to contain or not contain an animal based on this sum exceeding a predefined threshold. The required addition operations are much faster than the calculation of contours in the alternative approach.

4.2.3. H.264 motion analysis

An approach that was not seen in the literature is using the motion frames from the H.264 camera video stream. This encoded video stream is often used to transmit or store videos efficiently as the differences between frames are used in addition to full image frames. This combination means fewer full image frames need to be sent, reducing bandwidth requirements. The processing to generate the motion frames takes place in the camera's hardware and so has no additional overhead. Therefore this method is faster from the outset. It also uses less memory as past vectors do not need to be considered and the vector representation for a frame is also much smaller than the frame itself.

The third-party PiCamera library we use in our Python implementation offers a simple way of interacting with the camera module. One feature offered is the use of multiple camera ports, each with its own data stream. By setting one stream to provide the motion frames and another to provide the decoded JPEG frames, a similar result to using a custom decoder is achieved. The benefit of utilising two streams is that the complexity of decoding has been wrapped in a library, which is

[3] https://gitlab.dynaikon.com/dynaikontrap/vid2frames

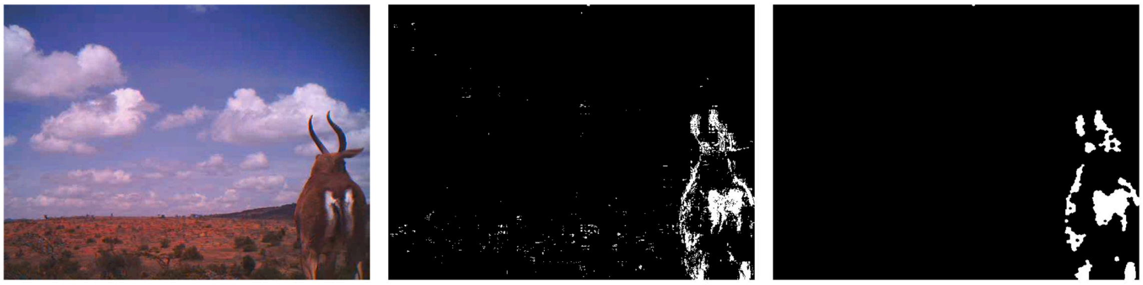

(a) Original image frame
(b) Raw MOG2 mask with visible noise
(c) Modified mask with noise removed

Fig. 3. The MOG2 method and morphological operations applied to a still image from a video in the WCS dataset.

assumed to be efficient. We have written code for synchronising these two streams, to ensure any detected movement leads to the correct image frame being passed on through the pipeline. Below we detail various criteria that could be used in the triggering process based on motion vectors.

4.2.3.1. Sum of Vectors (SoV). Here, the motion vectors are simply added and the magnitude of the result is taken. As the method results in a single value, only one threshold for motion detection needs to be applied. Appropriate thresholds can be determined by considering pixel size, animal size, distance, and focal length of the camera as discussed in Section 4.3. Fig. 4 visualises this approach.

A benefit of this approach is that noise or randomness in the movement vectors would be expected to cancel out, which is a simple, implicit method of checking the correlation of vectors: random movement of vegetation and other background objects will not lead to the camera being triggered. Note that it is possible, though unlikely, that two animals moving towards each other see their respective movement cancelled out.

4.2.3.2. Count of Thresholded Magnitudes (CoTM). This counts how many movement vectors have a magnitude above a threshold that is designed to eliminate random, small movements. A further threshold is applied to the count to determine whether or not movement is detected. A benefit of this approach is that many areas of small movement can be easily removed. However, this comes at the cost of determining a second threshold.

4.2.3.3. Sum of Thresholded Vectors (SoTV). Similarly to CoTM, noisy, small movement vectors are filtered out, but the remaining vectors are added up and the magnitude of the sum is compared against a threshold for movement.

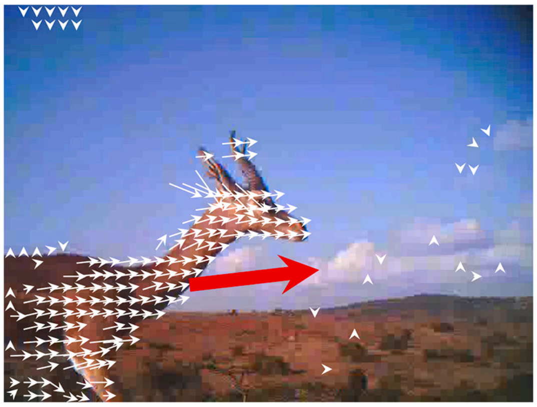


Fig. 4. Our SoV approach for determining motion on a video from the WCS dataset. The raw non-zero motion vectors are white, while their sum is illustrated as the large red vector. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

A comparison of these methods found the SoV approach to be the most effective, which is the one used in our presented results.

4.3. Defining thresholds

Each movement detector requires some form of threshold to be set. We utilise the particular attributes of the deployed lens; note that this makes evaluation of our method on externally obtained video sequences from undocumented lenses harder as these parameters need to be estimated. The fundamental optical equation is

$$\frac{s}{d} = \frac{pu}{f}, \tag{1}$$

where s is the real size of the subject, d is its distance from the camera, p is the number of pixels and u is the unit size of one pixel,[4] the product pu is the size of the subject on the camera sensor, and f is the focal length of the lens. This allows for a conversion between real size s and number of pixels p.

As an example, an animal with a visible area of 1 m by 0.5 m at a distance of 1 m from the camera with focal length of 3.6 mm results in pixel dimensions of 635 px by 317 px, a total of $2 \cdot 10^5$ pixels. The threshold of a background subtractor as in Section 4.2.2 for the number of white pixels would need to be set below that value to enable detection.

The motion vectors in Section 4.2.3 are projections onto the camera sensor plane. To determine the motion thresholds for *average* motion vectors, Eq. (1) can be modified to use the speed $v = \Delta s/\Delta t$ of the subject, where Δs is the real size of the displacement and Δt is the time between capturing frames. Substituting Δs for s into (1) gives

$$\frac{v}{d} = \frac{\Delta p}{\Delta t} \cdot \frac{u}{f}, \tag{2}$$

where Δp is now the average motion vector in pixels. Eq. (2) can be used to determine the lowest detectable speed: with $\Delta t = \frac{1}{25}$ s and $\Delta p = 1$ we get $v/d \approx 0.039\ \text{s}^{-1}$. An animal that is 1 m away from the camera would need to move at least with 39 mm/s ≈ 0.14 km/h for detection to be possible. Trigger thresholds can be set higher than this.

Our SoV approach does not use an average speed threshold as defined above, but instead a hybrid of this and the expected animal area. As such, a large subject moving slower than the threshold could still be treated as motion.

4.4. Smoothing

We compute motion characteristics independently for each frame, which can lead to relatively large frame-to-frame deviations or spurious vectors, see Fig. 5. Clearly, consecutive frames are not independent; we

[4] For our RPi camera $u = 1.4\,\mu$m; however, if a 640 × 480 video is recorded (scaled down by 4) then $u = 5.6\,\mu$m.

deploy an Infinite Impulse Response (IIR) filter, primarily for its low delay.[5] The IIR needs to be configured as a low-pass filter with little or no passband ripple and have a fast roll-off. These criteria are met by the Chebyshev type 2 filter (Smith, 2002, Chapter 20). Based on experimentation, we chose a filter order of three and a stop-band attenuation of −35 dB. We set the cut-off frequency from the user requirement for the expected animal speed from which we estimate, with the help of Eq. (2), the expected duration of an animal sequence crossing the frame: the cut-off frequency is set to the reciprocal value of the duration.

We apply the IIR smoothing to the raw movement vector components, and then proceed with the Boolean threshold that triggers the movement detection. Fig. 5 illustrates how thresholding the smoothed SoV output helps avoid false positive triggers.

4.5. *Timings*

Table 1 shows the execution time per frame for each of our methods. It supports our choice of using motion vectors over other methods. In our architecture, each frame of the video needs checking for motion, and any processing time over 40 ms limits the frame rate to below 25 fps. Neither a difference method, including the more evolved Zilong variation, nor a background subtraction method, can be used on an RPi for this reason.

5. Animal detection

In the proposed system, animal detection is set as the second and final filter on incoming frames. As the preceding movement-based filter has already been applied, the stream being passed to the animal detector is at a lower frame rate than that of the camera module. Thus, the animal detector is afforded more time to analyse each frame making the use of ML possible at this stage.

5.1. *Background*

Even if preceded by a motion filter, ML-based animal detection remains a time-critical process, such that any chosen model must be optimised for efficiency.

A good starting point is to look at image classifiers. These are systems that are given an image and predict what the image is of, giving a confidence for each prediction. A popular such architecture is the MobileNet (Howard et al., 2017) family. The newest iteration,

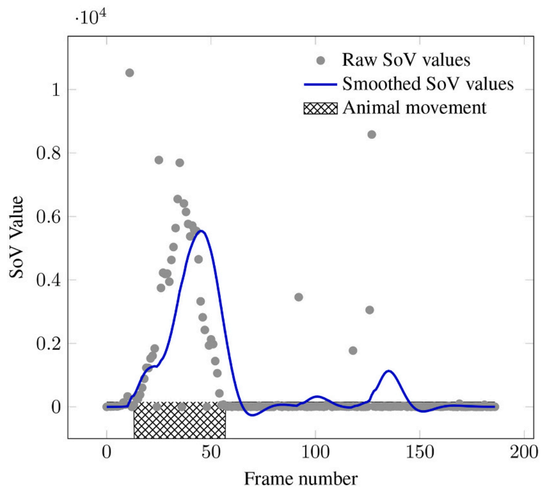


Fig. 5. Example of IIR smoothing of SoV motion filtering.

[5] Referring to execution time – there is a phase shift of a few frames present.

Table 1
Typical computation time per frame on an RPi.

Method	Time
Difference	55.9 ms
MOG2	256 ms
Motion vector	0.378 ms
Smoothed motion vector	0.397 ms

MobileNetv3 (Howard et al., 2019), is purported to run on Google Pixel-3 mobile phone CPUs at up to 11.7 ms per image. These classifiers are often able to run very quickly and work well for subjects that take up most of the image. When subjects are smaller and not the focus of an image, classifiers tend to perform worse. This makes them inappropriate for our use-case as there may well be many scenes with small animals at the edge of the image frame.

Object detectors are more suited to these situations, especially when the image contains many potentially interesting objects. In recent years there have been many advances in this field, although most approaches still assume vast computational resources. A promising architecture is YOLOv4 (Bochkovskiy et al., 2020). It is a single-stage object detector with relatively low computational requirements, whilst achieving competitive performance when compared to the state of the art. A more recent addition to the world of efficient object detectors is EfficientDet (Tan et al., 2020). One of this architecture's unique points is its scalability, allowing the same architecture to be used in high-power and low-power computing applications.

Another ML architecture of interest is Context R-CNN (Beery et al., 2020), which bases object detection predictions for a given frame on other historic frames. This is achieved by using a curated memory bank of frames for a given camera trap, allowing periodic activities, such as animals using the same path every day, to improve detection performance on a frame-by-frame basis. The technique is very well suited to the camera trapping domain. The architecture relies on any two-stage neural network-based detector architecture; they use a Faster R-CNN (Ren et al., 2015) model as a basis.

Irrespective of the model choice, it needs to be trained using camera trap data. It makes sense to train the model with images of species it is likely to encounter once deployed and as such a final choice of training data is reliant on the use-case. However, two popular datasets are Snapshot Serengeti and WCS Camera Traps,[6] both of which provide bounding box information for a subset of the dataset. We have chosen the latter for its diversity in species and geographical locations represented.

5.2. *Selected detection method*

5.2.1. *Architecture*

We have chosen YOLOv4 as the detection method. This model architecture has a number of variants (Wang et al., 2021). We opted for the lighter variant, a.k.a. “YOLOv4-tiny”, in the pipeline for its significant speed advantage, which, experimentally, we found to be a factor of approximately 7.7. The remainder of this paper will refer to YOLOv4-tiny simply as YOLOv4. This architecture requires input images of size 416 by 416 pixels, and so input camera frames are scaled down (squashed) before being processed. The modular design of the system allows for any other model architecture to be used, and benefit from future progress in the state of the art.

Image frames are passed to the model frame-by-frame for inferencing. The model then returns a list of predictions, each giving the dimensions of a bounding box and a confidence that an animal has been detected in the box. These predictions are then iterated through with an

[6] WCS Camera Traps dataset published by LILA BC at http://lila.science/datasets/wcscameratraps

animal detection being declared if the maximum box confidence exceeds a preset threshold. On the RPi the inference takes around 1 s. Note that an input frame may contain many animals, in which case multiple detections would be predicted.

5.2.2. Training

The official, MS-COCO pre-trained, version of this architecture has then been further trained, a technique known as transfer-learning, using the WCS dataset and its recommended data split, giving a credible *mean average precision* mAP_{50} score of 0.75. The mAP metric, used to evaluate the PASCAL VOC challenge dataset (Everingham et al., 2010), takes the *mean* over all considered species of their respective, species-specific average precision as defined below. The subscript 50 refers to the requirement that the area of the intersection of ground truth bounding box and predicted bounding box needs to be at least 50% of the area of the union of both for an otherwise true species identification to be considered correct. The network does not output a binary value for species identification but instead a belief $b \in [0,1]$ that a particular species has been identified. Thresholding these numbers at different values of t generates a precision-recall relationship $r \mapsto p(r) = \max_{\{t|Rt \geq r\}} P_t$ per species. The *average precision* is defined as the species-specific average over r of $p(r)$. Here, precision P_t is the proportion of correctly predicted images for the considered species amongst the predicted species images (those with $b \geq t$), while recall R_t is the proportion of species images that were correctly predicted. The WCS dataset contains species-specific annotations. During training we replace all species names with a single class of *animal* making training simpler. Training was undertaken using Darknet[7] as per default values, yielding configuration and weight files for deployment on the RPi.

5.3. Continuity and extrapolation

YOLOv4's output is discontinuous in time as animal predictions are not made for every frame in a motion event where an animal is present. In addition, the confidence level corresponding to an animal is inconsistent within a motion event and can be affected by many factors, including motion blur, lighting quality, and concealment. An approach is therefore needed to even-out gaps in predictions of animal presence. We have devised an algorithm that not only removes unlikely gaps in animal predictions, but also reduces the number of times the animal detector needs to be run. This algorithm looks both forwards and backwards in time, within a motion sequence, to extrapolate animal predictions efficiently and effectively.

5.3.1. Motion queue

To facilitate looking forward in time as well as backwards, the frames need to be buffered in some way. A *motion queue* has been devised for this purpose to act as an intermediary between the motion and animal filtering stages. Frames allowed to pass by the motion filter are placed on the motion queue: they are appended to a sequence of motion, which ends once a frame with insufficient motion is found. To prevent excessive delay in subsequent stages of the filtering pipeline, such motion sequences are capped to a maximum length that can be defined by the user. Each complete sequence of motion is then passed to the animal detector in accordance with the algorithm described next.

5.3.2. Algorithm

The goal here is to close gaps in time in the animal predictions made by YOLOv4. As will be discussed later, the algorithm yields an important secondary benefit in the form of improved speed performance. The underlying base assumption is that an animal is unlikely to be present in a given keyframe but not present in the preceding and succeeding frames. In this context a keyframe is representative of any animal that exists in the motion sequence. It is assumed that a prediction by the object detection model, with a confidence above a preset threshold, can be regarded as the truth. As such the positive detection must be due to a number of frames that are on either side of the keyframe. The number of frames to which this assumption may be applied can be calculated in a similar fashion to the methodology explained in Section 4.4. This process is referred to as *extrapolation* as it converts a single isolated prediction to a sequence of predictions. As the keyframe's actual position within the animal event may not be centred, it is possible that this extrapolation needs to be applied entirely to frames on a single side of the keyframe, and therefore we extrapolate in both directions.

Applying the above method, one quickly realises that there are multiple opportunities for improving efficiency. Firstly, if a frame has been labelled with the extrapolation process it does not need to be checked explicitly. Take an example situation where a motion sequence contains nine frames and extrapolation is performed one frame either side of a keyframe. This has been visualised in Fig. 6. The example shows that a best case occurs when only the frames at t_1, t_4, and t_7 are passed to the animal detector, so only three frames are analysed rather than nine. The animal detector processing time is therefore reduced by a factor of three in this example.

The above example indicates that one should prioritise in which order to analyse frames. This simple exercise is confounded slightly by the previously mentioned sporadic nature of animal detections. The example assumes the frames at t_1, t_4, and t_7 are all able to produce animal predictions by the animal detector, but this is not a safe assumption to make. Instead, another heuristic is needed to assign a priority to each frame in the sequence. A simple approach consists of using the motion score from the motion filter to indicate priority. The intuition here is that when the highest amount of motion is detected, the largest portion of the animal should be in view. Subsequently, the animal detector is assumed to have the best chance of predicting an animal.

Summarising, consecutive frames returned by the motion filtering step are put into a motion queue – a queue of motion sequences. Each motion sequence is separately analysed. Frames with highest magnitude are evaluated by the animal detection filter first and if positive, neighbouring frames[8] are declared interesting frames, which is the extrapolation. Once all frames in a motion sequence have been analysed, the interesting frames are returned from the motion queue in time order. These frames are later concatenated to form a compact video of the observation.

6. Results

6.1. Effectiveness measure

The task for the camera trap can be regarded as a filtering exercise, where the camera removes empty images and allows animal images to pass through.

We have two particular, but differing, use cases in mind. First is the case where there is a low bandwidth, low storage space or high cost of transmitting data, e.g., via satellite link. In this situation the user will want a focus to be placed on only transmitting animal images. Second is the case of a biodiversity study that may require every possible sighting of an animal to be logged and so the priority shifts to not missing anything potentially of interest. The ideal metric has a parameter that can shift focus from one use case to the other.

One widely used metric is the F_β-score which is derived from van Rijsbergen's (1979) Information Retrieval effectiveness measure, which in turn uses the weighted harmonic mean of *precision* (the proportion of kept frames that contain animals) and *recall* (the proportion of animal

[7] *Darknet* is the framework under which YOLOv4 is built. Code for this is at https://github.com/AlexeyAB/darknet, accessed 2020-11-26.

[8] The number of neighbouring frames declared as interesting is derived from user settings.

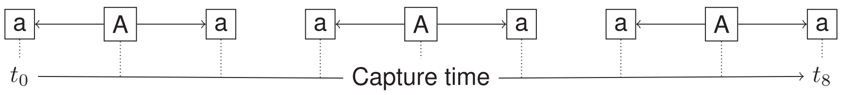


Fig. 6. Visualisation of a sequence of frames being declared to contain animals, using our algorithm. **A** represents a keyframe with an actual animal detection and **a** is a frame where animal presence has been inferred from a neighbouring keyframe. The animal detector needs only to be run three times to declare all frames in the motion sequence – from t_0 through to t_8 – as animals.

frames that were kept); the parameter β mediates between precision ($\beta = 0$) and recall ($\beta \to \infty$) while $\beta = 1$ gives equal weight to both.

While recall fits the second use case spot on, precision is *not* perfect for the first use case: it does not consider how well empty frames are being recognised as such.[9] We use *specificity* instead: the proportion of correctly recognised empty frames with respect to all empty frames. In other disciplines, specificity is also known as true negative rate, while recall is known as true positive rate. We thus introduce an ***ecology effectiveness*** measure:

$$E_\alpha = \left(\frac{\alpha}{S} + \frac{1-\alpha}{R}\right)^{-1} \in (0,1] \tag{3}$$

as the weighted harmonic mean between specificity S and recall R with $\alpha \in [0,1]$ and $S, R \in (0,1]$. E_α reduces to R for $\alpha = 0$ and to S for $\alpha = 1$ while $E_{0.5}$ considers specificity and recall with equal weight. In the case when S or R are zero, the definition of E_α should be extended to yield zero. As with most performance metrics, higher means better. To the best of our knowledge this measure is new and representative of the ecology use cases in our application.

Another measure that is sometimes used for classification is the Matthews Correlation Coefficient (MCC), which is the geometric mean between informedness (defined as $S + R - 1$) and markedness (Powers and Ailab, 2011). Although the informedness aspect captures some of our objectives, MCC is a single number that does not allow us to mediate between our two use cases.

6.2. Benchmarking setup

There are two ways in which the metric can be applied to test the system. Firstly, as a comparison with existing camera traps. Secondly, to evaluate the pipeline with respect to the underlying neural network. As there is no dataset available that contains full videos with corresponding PIR sensor information, such data would need to be manually produced. This was done at a small scale to inform development, but it was not feasible to scale this up for a robust test set. We instead focus on benchmarking different system configurations compared to existing ML approaches.

6.2.1. Data

The GBIF database[10] contains over 2300 videos of animals, although many of these are not suitable for our purpose. We have curated a subset of 330 videos that are shot with static cameras. These videos are therefore good representations of the style of input this pipeline is expected to receive. Every frame from each of these videos has been manually annotated, indicating whether or not it contains an animal. We have provided the training (30) and validation (300) split as used to tune parameters and evaluate the system, respectively. It is important to note that camera-related parameters have not been annotated or used in testing the DynAIkonTrap. For these tests we have assumed the focal length for all cameras to be the same as that of the RPi camera module – this is unlikely to be true. It is therefore likely that users can expect improved performance if they have better knowledge of the camera hardware used and the deployment scenario.

6.2.2. Test bench

Each video from our dataset is split into image and motion vector frames using our vid2frames library. When testing the DynAIkonTrap pipeline the camera input can be replaced by a class that feeds the prepared frames into the pipeline. Any model that is run as a comparison can be fed the image frames with the motion being discarded. This means all models receive identical input data.

Each model will then indicate if it has detected an animal in a frame, allowing recall, specificity and the effectiveness measure E_α to be determined from the ground truth. These numbers are computed on the full set of validation videos, treating them as one concatenated video. As some videos will contain no animals and others an animal in every frame, it is not useful to determine E_α on a video-by-video level. In addition to this, the test bench keeps track of how many times the underlying ML model – the limiting factor with respect to processing speed – was run.

6.3. Results

The curated set of videos has been passed through a variety of configurations of the DynAIkonTrap pipeline, as well as the underlying YOLOv4 model itself. These configurations with their effectiveness measures of $E_{0.5}$ and average $\overline{E_\alpha}$ are summarised in Table 2. It is clear that using any of these approaches results in significant speed improvements, in the form of fewer YOLOv4 runs. YOLOv4 takes close to 1.2 s inference time per frame on an RPi. In comparison, the movement analysis takes less than 0.4 ms per frame making the YOLOv4 model the major bottleneck. The reduction in the number of model runs therefore leads to significantly improved processing speed of the video data set.

Using Table 2 it is possible to determine the rate at which frames can be processed and therefore the maximum number of frames each configuration can process in a day. The processing frame rates have been calculated assuming the animal detector alone (Table 2: B) runs at 0.83 fps $\left(\frac{1}{1.2\,\mathrm{s}}\right)$, and other delays are ignored. The duration per day is the number of frames that the animal detector can process in a day converted to a time, assuming data is recorded at 20 fps. More can be recorded per day, but it would have to be processed the next day.

As a major application of this system is low-power devices, it is useful to determine which pipeline configuration is the most power-efficient. The power consumption can be split into the following components: system idle (2.20 W) – when our software is not running; motion filtering (additional 1.55 W) – the pipeline idle state; and animal filter (additional 2.50 W) – due to the animal detection model running,[11] which adds up to 6.25 W. The only component of power consumption we have control over with the different pipeline configurations is the number of animal filter runs. To indicate how this maps to a real-world deployment we have determined an average power consumption for the dataset. To do this, the animal detector power consumption is multiplied by the proportion of dataset frames that are run through the animal detection model. Note that the peak power consumption will be the

[9] In fact, Information Retrieval has no interest in being able to spot irrelevant documents (which correspond to empty frames here): virtually *every* of the billions of documents in a collection is irrelevant given a query. Information Retrieval metrics therefore normally do not consider true negatives.

[10] https://www.gbif.org/

[11] Power values given for RPi 4B

Table 2
Results of different trap configurations on the validation subset of the corpus; note that (B) is a baseline for comparison against Machine-Learning only.

DynAIkonTrap Configuration	Motion filter	Motion queue	Animal filter	$E_{0.5}$	$\overline{E_\alpha}$	Model runs (% saving)
(A) Motion only	✓	✘	✘	0.485	0.488	–
(B) Animal only (YOLOv4)	✘	✘	✓	0.611	0.640	99,727 (±0%)
(C) Motion queue + animal	✘	✘	✓	**0.731**	**0.731**	46,184 (−53.7%)
(D) Motion + animal	✓	✘	✓	0.400	0.459	43,116 (−56.8%)
(E) Motion + motion queue + animal	✓	✓	✓	0.481	0.515	**15,824 (−84.1%)**

same across all configurations, the average can come down as fewer animal detector runs are needed (Table 2: C, D, E), and the best power-performance is achieved when there are fewer animal frames. The power consumption due to the animal detector can therefore drop to 0 W so only the idle power consumption of 3.75 W (2.20 W+1.55 W) remains, at which point the focus shifts to reducing the hardware power consumption. We have successfully managed to port the DynAIkonTrap concept to an RPi Zero in efforts towards addressing this.

The derived results are summarised in Table 3 and attribute further meaning to the reduction in animal detector runs from Table 2. Using the configuration with all three stages therefore has significant positive implications on both the processing capacity and power usage of the system.

Fig. 7 plots each method's resulting E_α over the full $\alpha \in [0, 1]$ range to indicate the trade-off between recall ($\alpha = 0$, lhs) and specificity ($\alpha = 1$, rhs). The shape of the motion only approach (Table 2: A) is slightly undesirable in that it has a fairly low recall. An ideal motion filter would have a very high recall, with a potentially lower specificity, to match the level of intelligence of the filter. It is undesirable to filter too excessively at this stage as only a proxy for animal presence (motion) is used. As mentioned, the low performance here may be related to the test setup, where exact parameters of the hardware used to generate the videos were not known, thereby not allowing a properly tuned system.

By adding the motion queue to the animal filter (C) the performance becomes almost constant with respect to α. The results demonstrate that the configuration is able to counteract our main criticism of YOLOv4 for analysing videos: regarding every frame as temporally isolated. By applying the animal prediction extrapolation (Section 5.3) we are therefore able to improve recall without greatly sacrificing specificity.

Combining only the motion filter and animal filter (D) leads to poor performance, unless the user is exceptionally focused on not outputting empty frames.

Using all stages in combination (E) yields a plot similar in shape to the animal-only approach (B), but shifted down. Most likely this is due to the motion filtering stage's lower performance. The configuration, however, brings a reduction in the number of model runs by 84.3%, which has associated benefits in power efficiency and processing capacity.

Multiple multi-day deployments of various versions of the software on RPis have also demonstrated the full architecture (including hardware) to produce expected output such as Fig. 8. The software is under continuous improvement to enhance and bring innovation to the system,

Table 3
Characteristics for each configuration, derived from Table 2. *Processing rate* – rate at which frames are processed; *Duration per day* – average duration of animal presence that can be analysed per day; *Average animal detector power* – average power usage over the dataset due to the animal filter, based on a real-world 2.5 W power usage for the animal filtering stage on an rpi 4B. Column headings correspond to rows in Table 2. *As there is no pre-filtering for (B), the 0.83 fps rate means 1 h' worth of frames can only be processed with frames spaced at 1 frame every 1.2 s.

	(B)*	(C)	(D)	(E)
Processing rate/fps	0.83	1.80	1.93	**5.24**
Duration per day/h	1.00	2.16	2.31	**6.29**
Average animal detector power/W	2.50	1.16	1.08	**0.40**

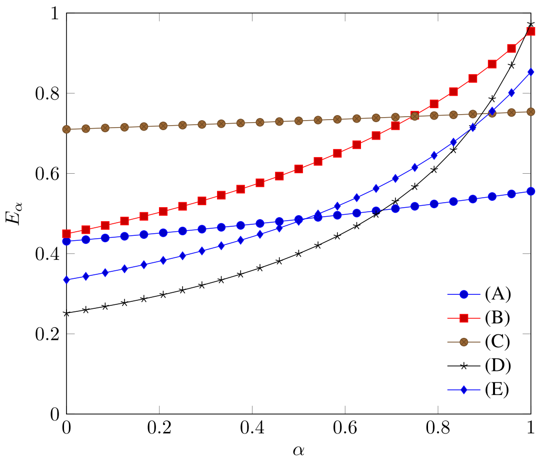


Fig. 7. Plots of E_α over full α range for different methods.

such as adding a human detector to enable privacy protection.

7. Discussion

We have presented an innovative solution to many problems with traditional camera traps. This thought-to-be novel architecture moves away from the tradition of using motion sensors in camera traps and demonstrates the success of a camera-only system.

We introduced a camera trap system that operates solely on camera video, with no additional sensors used to trigger the capturing process. The use of ML for near-real-time animal detection on low-power devices is made possible by our fast movement detection mechanism. As part of the evaluation, we have contributed an annotated data set for video camera trap evaluation. We have also devised a novel effectiveness measure for camera traps in the ecology domain (Eq. (3)). The resulting DynAIkonTrap is highly configurable, allowing users to tune the system to experimental requirements. As the pipeline is completely implemented in software – there is no initial hardware PIR trigger – it is possible to completely reconfigure the camera trap remotely without altering any hardware. Finally, we have demonstrated that the system works through an evaluation on pre-recorded videos.

7.1. *Ideas for the future*

7.1.1. *Long-distance*

A concept not explored in this paper is long-distance camera trapping. This would consist of using a telephoto lens. Traditional camera traps are not able to perform long-distance recording as the PIR sensor would need to be too sensitive to be a useful trigger. This is not a limit for our camera trap system as it relies on what is visible by the actual camera. As such, it can record over any distance that can be seen with a camera lens. It would be valuable to explore this concept as it may prove to be useful for camera trapping in locations that are difficult to access, but observable from a distance.

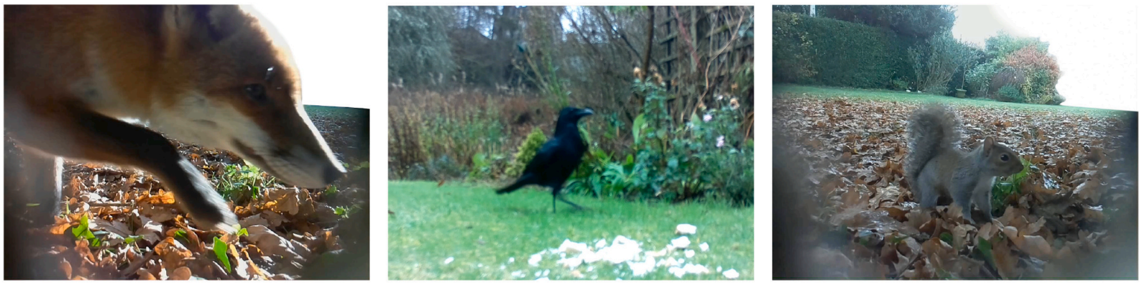


Fig. 8. Example stills from videos captured with various versions of a DynAIkonTrap. Parts of the background have been manually removed for privacy concerns (left and right stills).

7.1.2. Using time in the neural network

The animal detection model used in this project ultimately performs frame-wise inference, meaning temporal information is not used in the ML model. It would be worth exploring possible methods for improving animal detection performance by using temporal information in the neural network. As motion vectors are extracted and used elsewhere in the pipeline, it would be easy to feed these into the neural network, making this an attractive approach as work is only needed on the ML model. A slightly more complex solution could be to pass multiple consecutive frames to the neural network as an alternative, or in addition to motion vectors. Efficient implementations of Context R-CNN (Beery et al., 2020) mentioned in Section 5.1 could also be of interest for this purpose.

7.1.3. Bespoke neural networks for species sets

While we selected a suitable neural network for species identification, we spent little research so far to train this network to sets of user-selected or prevalent species in the region where the camera trap is deployed. Training for a bespoke set of species is both possible and desirable. Further research is needed to identify and automate a process that, given a set of species of interest, outputs a robust and reliable neural network model with suitable weights.

7.1.4. Low power neural networks in hardware

It is desirable to reduce the power consumption of the DynAIkonTrap to make deployment in remote locations easier and less expensive. An implementation by Si et al. (2020) of a Multilayer Perceptron model on an Field Programmable Gate Array (FPGA) was shown to run in a similar time frame to a software-based approach, but at a clock speed lower by a factor of 144. The authors portray this as the ability to achieve a lower power consumption or improve performance by a factor of 144. Results published by Intel (2016) indicate a Convolutional Neural Network can be run on an FPGA with twice the efficiency of a software/CPU implementation.

It is clear that hardware implementations of neural networks offer improved power efficiency over software-based alternatives and their use has the potential to reduce the overall system's power consumption. As this idea makes use of programmable hardware, any neural network implementation is not permanent and so the customisability of the system is not sacrificed. It may also be worthwhile investigating the potential efficiency gains of using FPGAs in the motion filtering stage. Ultimately, it could be possible to implement the majority of the system with FPGAs, so a much lower power processor could be used for remaining tasks such as networking and human interaction.

8. Conclusion

The DynAIkonTrap project demonstrates improved performance of a camera-only RPi system over the traditional PIR sensor-based alternative. In doing so, a novel video filtering pipeline has been introduced that facilitates the running of an animal detection ML model on a live video stream. The minimum configuration of the DynAIkonTrap can be very compact and costs less than the identified commercially available camera traps, making it affordable for citizen scientists. The open-source, modular, and hackable nature of this project provides a great opportunity for further work and improvements to the system.

Data, software, and conflicts

We provide the open-source code for the camera trap, documentation, video evaluation corpus, and our evaluation library at https://dynaikon.com/trap. We declare that we have no conflicts of interest and, except for DynAIkon, we do not have a relation with any of the brands or companies mentioned in this article.

Author contributions

MR led the design, development, and evaluation of the system in close consultation with, and supervision of, SR and FFL. Code reviews were conducted by RG and SR. Further feedback during research and development was provided by KW on neural learning and by RR on documentation, workflow, and dissemination. All authors contributed critically to the drafts and gave final approval for publication.

Declaration of Competing Interest

None.

Acknowledgements

This research is supported in part by the project COS4CLOUD, which has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 863463. COS4CLOUD is a project coordinated by the Institut de Ciències del Mar ICM-CSIC, based in Barcelona (Spain).

References

Beery, S., Wu, G., Rathod, V., Votel, R., Huang, J., 2020. Context R-CNN: Long term temporal context for per-camera object detection. In: Conference on Computer Vision and Pattern Recognition (CVPR). IEEE Computer Society, pp. 13072–13082. https://doi.org/10.1109/CVPR42600.2020.01309.

Bochkovskiy, A., Wang, C.Y., Liao, H.Y.M., 2020. Yolov4: Optimal Speed and Accuracy of Object Detection arXiv:2004.10934.

Driessen, M.M., Jarman, P.J., Troy, S., Callander, S., 2017. Animal detections vary among commonly used camera trap models. Wildl. Res. 44, 291. https://doi.org/10.1071/wr16228.

Droissart, V., Azandi, L., Onguene, E.R., Savignac, M., Smith, T.B., Deblauwe, V., 2021. PICT: a low-cost, modular, open-source camera trap system to study plant–insect interactions. Methods Ecol. Evol. 12, 1389–1396. https://doi.org/10.1111/2041-210X.13618.

Everingham, M., Van Gool, L., Williams, C.K., Winn, J., Zisserman, A., 2010. The PASCAL visual object classes (VOC) challenge. Int. J. Comput. Vis. 88, 303–338.

Findlay, M.A., Briers, R.A., White, P.J.C., 2020. Component processes of detection probability in camera-trap studies: understanding the occurrence of false-negatives. Mammal Res. 65, 167–180. https://doi.org/10.1007/s13364-020-00478-y.

Glen, A.S., Cockburn, S., Nichols, M., Ekanayake, J., Warburton, B., 2013. Optimising camera traps for monitoring small mammals. PLoS One 8. https://doi.org/10.1371/journal.pone.0067940.

Hobbs, M.T., Brehme, C.S., 2017. An improved camera trap for amphibians, reptiles, small mammals, and large invertebrates. PLoS One 12, e0185026. https://doi.org/10.1371/journal.pone.0185026.

Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., Adam, H., 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications arXiv e-prints arXiv:1704.04861.

Howard, A., Sandler, M., Chen, B., Wang, W., Chen, L., Tan, M., Chu, G., Vasudevan, V., Zhu, Y., Pang, R., Adam, H., Le, Q., 2019. Searching for MobileNetV3. In: International Conference on Computer Vision, pp. 1314–1324.

Intel, 2016. Efficient Implementation of Neural Network Systems Built on FPGAs, and Programmed with OpenCL™. https://www.intel.co.uk/content/dam/www/programmable/us/en/pdfs/literature/solution-sheets/efficient_neural_networks.pdf.

Jolles, J.W., 2021. Broad-scale applications of the raspberry pi: a review and guide for biologists. Methods Ecol. Evol. 12, 1–18. https://doi.org/10.1111/2041-210X.13652.

Jumeau, J., Petrod, L., Handrich, Y., 2017. A comparison of camera trap and permanent recording video camera efficiency in wildlife underpasses. Ecol. Evol. 7, 7399–7407. https://doi.org/10.1002/ece3.3149.

Klemens, J.A., Tripepi, M., McFoy, S.A., 2021. A motion-detection based camera trap for small nocturnal mammals with low latency and high signal-to-noise ratio. Methods Ecol. Evol. 12, 1323–1328. https://doi.org/10.1111/2041-210X.13607.

Ko, B.H., Jeong, S.G., Ahn, Y.G., Park, K.S., Park, N.C., Park, Y.P., 2014. Analysis of the correlation between acoustic noise and vibration generated by a multi-layer ceramic capacitor. Microsyst. Technol. 20, 1671–1677. https://doi.org/10.1007/s00542-014-2209-5.

McIntyre, T., Majelantle, T.L., Slip, D.J., Harcourt, R.G., 2020. Quantifying imperfect camera-trap detection probabilities: implications for density modelling. Wildl. Res. 47, 177. https://doi.org/10.1071/wr19040.

Meek, P.D., Ballard, G., Claridge, A., Kays, R., Moseby, K., O’Brien, T., O’Connell, A., Sanderson, J., Swann, D.E., Tobler, M., Townsend, S., 2014a. Recommended guiding principles for reporting on camera trapping research. Biodivers. Conserv. 23, 2321–2343. https://doi.org/10.1007/s10531-014-0712-8.

Meek, P.D., Ballard, G.A., Fleming, P.J.S., Schaefer, M., Williams, W., Falzon, G., 2014b. Camera traps can be heard and seen by animals. PLoS One 9, e110832. https://doi.org/10.1371/journal.pone.0110832.

Meek, P., Ballard, G., Fleming, P., Falzon, G., 2016. Are we getting the full picture? Animal responses to camera traps and implications for predator studies. Ecol. Evol. 6, 3216–3225. https://doi.org/10.1002/ece3.2111.

Nazir, S., Newey, S., Irvine, R.J., Verdicchio, F., Davidson, P., Fairhurst, G., van der Wal, R., 2017. WiseEye: next generation expandable and programmable camera trap platform for wildlife research. PLoS One 12. https://doi.org/10.1371/journal.pone.0169758.

Petso, T., Jamisola Jr., Rodrigo S., Mpoeleng, D., 2022. Review on methods used for wildlife species and individual identification. Eur. J. Wildl. Res. 68 https://doi.org/10.1007/s10344-021-01549-4.

Powers, D., Ailab, 2011. Evaluation: from precision, recall and F-measure to ROC, informedness, markedness & correlation. J. Mach. Learn. Technol. 2, 2229–3981. https://doi.org/10.9735/2229-3981.

Proppe, D.S., Pandit, M.M., Bridge, E.S., Jasperse, P., Holwerda, C., 2020. Semi-portable solar power to facilitate continuous operation of technology in the field. Methods Ecol. Evol. 11, 1388–1394. https://doi.org/10.1111/2041-210x.13456.

Ren, S., He, K., Girshick, R., Sun, J., 2015. Faster R-CNN: Towards real-time object detection with region proposal networks. In: Cortes, C., Lawrence, N., Lee, D., Sugiyama, M., Garnett, R. (Eds.), NeurIPS, pp. 91–99.

Robley, A., Gormley, A., Woodford, L., Lindeman, M., Whitehead, B., Albert, R., Bowd, M., Smith, A., 2010. Evaluation of Camera Trap Sampling Designs Used to Determine Change in Occupancy Rate and Abundance of Feral Cats. Arthur Rylah Institute for Environmental Research.

Schindler, F., Steinhage, V., 2021. Identification of animals and recognition of their actions in wildlife videos using deep learning techniques. Ecol. Inform. 61, 101215 https://doi.org/10.1016/j.ecoinf.2021.101215.

Si, J., Harris, S.L., Yfantis, E., 2020. Neural networks on an FPGA and hardware-friendly activation functions. J. Comput. Commun. 08, 251–277. https://doi.org/10.4236/jcc.2020.812021.

Smith, S.W., 2002. Digital Signal Processing: A Practical Guide for Engineers and Scientists. Elsevier.

Swanson, A., Kosmala, M., Lintott, C., Simpson, R., Smith, A., Packer, C., 2015. Snapshot Serengeti, high-frequency annotated camera trap images of 40 mammalian species in an African savanna. Sci. Data 2. https://doi.org/10.1038/sdata.2015.26.

Tabak, M.A., et al., 2019. Machine learning to classify animal species in camera trap images: applications in ecology. Methods Ecol. Evol. 10, 585–590. https://doi.org/10.1111/2041-210X.13120.

Taggart, P.L., Peacock, D.E., Fancourt, B.A., 2020. Camera trap flash-type does not influence the behaviour of feral cats (Felis catus). Aust. Mammal. 42, 220. https://doi.org/10.1071/am18056.

Tan, M., Pang, R., Le, Q.V., 2020. EfficientDet: Scalable and efficient object detection. In: Conference on Computer Vision and Pattern Recognition. IEEE. https://doi.org/10.1109/cvpr42600.2020.01079.

Trnovszký, T., Sýkora, P., Hudec, R., 2017. Comparison of background subtraction methods on near infra-red spectrum video sequences. Proc. Eng. 192, 887–892. https://doi.org/10.1016/j.proeng.2017.06.153.

Urbanek, R.E., Ferreira, H.J., Olfenbuttel, C., Dukes, C.G., Albers, G., 2019. See what you’ve been missing: an assessment of Reconyx® PC900 Hyperfire cameras. Wildl. Soc. Bull. 43, 630–638. https://doi.org/10.1002/wsb.1015.

van Rijsbergen, C.J., 1979. Information Retrieval, 2nd ed. Butterworth-Heinemann, USA.

Wang, C.Y., Bochkovskiy, A., Liao, H.Y.M., 2021. Scaled-YOLOv4: Scaling cross stage partial network. In: Conference on Computer Vision and Pattern Recognition (CVPR), pp. 13029–13038.

Wei, W., Luo, G., Ran, J., Li, J., 2020. Zilong: a tool to identify empty images in camera-trap data. Ecol. Inform. 55, 101021 https://doi.org/10.1016/j.ecoinf.2019.101021.

Weingarth, K., Zimmermann, F., Knauer, F., Heurich, M., 2013. Evaluation of six digital camera models for the use in capture-recapture sampling of Eurasian Lynx. Waldökologie Online 13, 87–92.

Xi, T., Wang, J., Qiao, H., Lin, C., Ji, L., 2021. Image filtering and labelling assistant (ifla): expediting the analysis of data obtained from camera traps. Ecol. Inform. 64, 101355 https://doi.org/10.1016/j.ecoinf.2021.101355.

Zhu, X., Wang, Y., Dai, J., Yuan, L., Wei, Y., 2017. Flow-guided feature aggregation for video object detection. In: Proceeding of the IEEE International Conference on Computer Vision (ICCV). Venice, Italy, pp. 408–417 arXiv:1703.10025.

Zivkovic, Z., 2004. Improved adaptive Gaussian mixture model for background subtraction. In: International Conference on Pattern Recognition. IEEE. https://doi.org/10.1109/icpr.2004.1333992.

Zivkovic, Z., van der Heijden, F., 2006. Efficient adaptive density estimation per image pixel for the task of background subtraction. Pattern Recogn. Lett. 27, 773–780. https://doi.org/10.1016/j.patrec.2005.11.005.