2011 International Conference on Environment Science and Engineering
IPCBEE vol.8 (2011) © (2011) IACSIT Press, Singapore

Automated real-time dynamic identification of flying and resting butterfly species in the natural environment

Kar Seng Loke, Simon Egerton
School of IT
Monash University Sunway Campus
Malaysia
loke.kar.seng@infotech.monash.edu.my
simon.egerton@infotech.monash.edu.my

David Cristofaro, Steven Clementson
School of IT
Monash University Sunway Campus
Malaysia
dtcri1@monash.edu
smcle2@monash.edu

***Abstract*—In this paper we present a system that can identify butterfly species in real-time in their natural environment. Our system uses image process techniques to extract butterfly key-points and an enhanced method of tree classification to learn and identify the butterflies. We have trained the system on 10 butterfly species and have performed real-time validation tests at the local butterfly park. Our results show robust performance with an average identification accuracy of 85%.**

Keywords biodiversity; image-processing; butterflies; real-time;

I. INTRODUCTION

Monitoring the state of the environment, ecosystems and biodiversity is essential to better understand the health of our planet. There are many key indicators used to evaluate the health of an ecosystem, some of which include the populations of specific fauna. One of these key indicators is moth (and butterfly) population levels [1]. The process of monitoring population levels first requires identification (taxonomy) of specific species which can be time-consuming and expensive, requiring countless man-hours by trained ecologists. Automating the identification process is highly desirable. Software can be used to identify and count specific species of animals. This data can then be used in a larger environmental model which is capable of monitoring the health of an ecosystem, such as is envisaged by the Automated Eye on Nature (AEON) project [2].

Much work has been done in the past on recognising human faces with computer vision. Recently the technology and techniques have been applied to identifying animals and differentiating between species [3]. Insects are particularly suitable in this regard as they often have rigid bodies. The wing patterns of bees have been successfully used to identify individual bees in a colony [4]. In this paper we describe our approach to automatically identify butterflies using image processing techniques and a refined classification technique. Butterflies are highly suited to identification because their wing patterns are colourful and highly variable between species.

II. RELATED WORKS

Research on automated identification species using computer-based vision have been demonstrated over a diverse set of fauna. Tilo Burghardt and Janko Calic perform detection and tracking of lion faces to extract and extrapolate lion locomotive activities [5] using the AdaBoost algorithm [6]. Tilo et al later expanded the computer vision for monitoring seabirds [7] and penguins [8]. Other researchers have worked on identifying individual sea turtles [9]. In a closely related research Mayo and Watson [10], and Watson et al [11], have worked on the automatic identification of live moths. However, in both these works, the moths are not in flight unlike in our model where we use butterflies in flight in their natural environment.

III. METHODOLOGY

A. Overview

The approach we develop in this paper requires the use of a static video camera to capture footage of butterflies in their natural environment. The butterfly species to be identified must first be trained separately using still images of the relevant butterfly species.

The training process aims to capture species specific features such as colour patterns that can be used to distinguish the various species. The location of the wing patterns that are used to discriminate the species specific patterns are called keypoints. The training and identification methods we use are adapted from Random Tree [12].

After training is complete, the system is applied to live video footage to identify trained species in real-time. However before the identification process can begin, we need to pick out the butterflies from the surrounding scene. We separate out the butterflies using a technique called background subtraction. The technique we use requires that the initial scene is devoid of any butterflies so that the background scene can be learned by the system.

B. Background Substraction

In order to be able to identify butterflies and other objects in the video, the background must first be separated from the foreground. This is a process known as background subtraction or segmentation. The desired output of this

process is a video of the object of interest with the background erased. This video can then be used in the identification processes, to detect the presence of any butterflies.

Background subtraction generally involves three stages: learning a background model, frame differencing and foreground clean up [13]. During an initial learning phase, the program is fed frames of video that contain no foreground objects (butterflies in this case). These frames are "learned" to be background and what appears in them is considered unimportant. This knowledge of background frames is the called the background model. Two methods of background modeling have been implemented in this work: the averaging method and the codebook method.

The second stage involves frame differencing. This is the practice of taking a current frame and comparing it with the background model. From this, a mask is generated (see fig. 1). The mask is a black and white image in which black regions indicate a match with the background and white regions represent foreground. The mask is the raw result of the background subtraction.

In the process of frame differencing, each pixel is compared to its codebook. Each codebook is a set of codebook entries. Each codebook entry is an upper and lower bound on colour intensity that the pixel may occupy as part of the background [14].

Figure 1. Background subtraction result using the codebook method

During the learning phase, each pixel of a new frame is compared to the previously learned codebook. If the pixel is within a particular threshold distance of a codebook entry, or within a codebook entry, that pixel is classified as background. Pixels outside this range are deemed to be foreground.

The YUV colour space was chosen for use, rather than RGB. The variation in a background predominantly occurs in the brightness; hence a colour space with a dedicated brightness channel offers more flexibility and accuracy. It allows a wider threshold to be set for the brightness channel, to allow for shadowing effects. The thresholds used were 20 for the Y channel and 10 for U and V channels. These values were determined through subjective experimentation with different video files.

Cleaning the mask is the final stage of background subtraction, using a connected component analysis algorithm, and helps considerably in producing an accurate output from the background subtraction. It removes noise in the mask and smoothes the edges around objects.

C. Keypoint Extraction

The keypoint extractor used by our butterfly detection algorithm is called YAPE (Yet Another Point Extractor), it belongs to the BazAR library and interfaces easily with OpenCV. Keypoint extraction is the process whereby the main feature points in an image are computed. These may be corners, edges, lines or dots for example. Keypoints are invariant to rotation, scale and viewing angle making them ideal features for detection [12]. Figure 2 below illustrate the 70 best keypoints found on two different views of the same butterfly – one after a random transform. Once the keypoints have been found, the patch centred on each keypoint is retrieved. The set of patches are required for the training and detection with Random Trees.

D. Random Trees

The random trees technique involves building and training the tree set before video processing begins to achieve fast runtime performance. The procedure can be split into three sections; building the trees, training the trees and using them to classify objects.

The tree structure implemented in our model was a simple binary one. Tree nodes are of two types: internal nodes and leaf nodes. The internal nodes of the tree contain a test that yields a binary result that allows the patch to move either to the left child or right child. The leaf nodes of the tree store a set of probabilities, one for each butterfly species (Table I, II).

The tests used in the internal nodes are random as this makes for much faster tree building than the classical method of generating tests that will split the training data in half at each node [12]. Three different types of tests were implemented in the program. The first test is the most basic, and simply computes the average colour intensity of one randomly chosen channel, and compares it to a randomly generated number between 0-255. If the average intensity is less than this value, the patch moves to the left child, if it is greater the patch moves to the right subtree. This method works best for small patch sizes (3x3) and is only able to distinguish butterflies of distinctly different colours.

The second type of test used compares the colour intensity of a channel at a single pixel location with the colour intensity of the same channel at another pixel location within the patch, as in Lepetit and Fua [12]. The colour channel and pixel locations are chosen randomly for each test. The final test type combines the two previous tests, by comparing the average intensities of two 2x2 sub-patches at randomly chosen locations. This type of test was found to be the most reliable; achieving the best leaf node probability distributions.

The tests were first implemented using a channel selected from the RGB colour space. This was later changed to the YUV colour space, so that the brightness of the patch could be ignored. This allowed for better matching when the training images were taken in different light conditions to the video.

IV. TRAINING AND IMPLEMENTATION

The matching algorithm should be invariant to rotation and skew. For example: an image of a butterfly should be matched regardless of its orientation or the angle that it is viewed. To produce this invariance, the random trees are trained with a number of randomly transformed images generated from the training images. These transformations combine translation, scaling, shearing and rotation.

For each transformation, the corners of the learning image are stretched by a random amount, up to 40%. This level of stretching generally shrinks the learning images to a size that is similar to the butterflies in the video footage, allowing for more accurate matching. Shearing has the effect of warping or skewing the image to a possible alternate view of the butterfly, as if it were being viewed from a different angle. These affine transformation are followed by a random rotation between -180 and 180 degrees. This angle range was chosen as is allows butterflies viewed from any direction to successfully match. The process of generating the transforms creates a larger set of training images that will produce a set of random trees not affected by differences in perspective.

The training images were transformed randomly enough times so that a total of 1000 keypoints were generated per butterfly class. If 50 keypoints were found on every image, this would mean each training image was transformed 4 times (as the original image was also used for training, not just the transforms). In practise, many images and their transforms yield less than 50 keypoints so more transforms would have been generated. This method was used opposed to generating a fixed number of random transforms because all classes of butterfly were required to have an equal number of keypoints stored in the trees to prevent skewing of the results towards larger or more patterned butterflies.

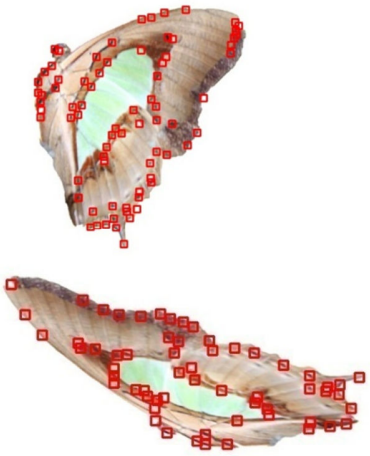

Figure 2. Keypoints found on the image before (left) and after a random transform (right).

The entire set of keypoint patches extracted from the training images and their random transforms are used to train the trees. Every patch is passed through every tree separately which results in long computation times, but this is done before video processing begins. Every patch belongs to one type of butterfly, and the leaf node in which the patch lands contains an array of counters, in addition to a total patch counter (Table I). When a patch lands in a leaf node, the counter for its corresponding class of butterfly is incremented, and the total patch count for that leaf node is incremented.

In any given leaf node, the ideal situation would be to have a single butterfly with a clear frequency bias. The greater the spread of frequencies, the less information the leaf node contains about the butterfly class.

TABLE I. EXAMPLE OF THE DATA STORED AT EACH LEAF NODE.

ID	0	1	2	3	4	5	6	7	8	9	10
Count	0	4	0	1	8	1	0	0	2	0	0
Tot Count	16										

It is desired to minimise the entropy of any given leaf node:

$$H(L) = -\sum_{i=0}^{i=n} p_i \log p_i$$

Where L is the leaf node probability distribution, there are n butterfly classes and p_i is the probability that the object is a match to the i^{th} class. The average entropy of the third test type was found to be the lowest and was the test we selected for our tests.

A. Object Classification

During runtime, each foreground object from the video is run through the keypoint generator and a set of patches is created. Each patch is passed through every tree and the accumulation of the results gives rise to a subsequent probability that the detected object belongs to each butterfly class. Table II shows a typical leaf node frequency distribution with computed probabilities.

TABLE II. PROBABILITY DISTRIBUTION AT LEAF NODE

ID	0	1	2	3	4	5	6	7	8	9	10
Prob	0	0.25	0	0.06	0.5	0.06	0	0	0.13	0	0

The probability distributions are accumulated for each patch and each tree to achieve an overall distribution. If the highest probability is greater than some threshold value, then the object is said to belong to that corresponding butterfly class. If all probability values are below the threshold then it is deemed that the object belongs to none of the butterfly classes. For a given input image with *m* keypoints, it matches butterfly *x* if:

$$\underset{x}{\operatorname{argmax}} \left[\frac{1}{mn} \sum_{k=0}^{k=m} \sum_{t=0}^{t=n} \frac{c_x^k}{c_t} \right]$$

Where c_x is the count of butterfly *x*, c_t is the total count in the leaf node in which the keypoint *m* lands, and there are *n*

trees. The argmax returns the argument *x* (the butterfly ID) when the value is maximum. However the max result is only valid if it equals or exceeds a matching threshold, Z. Simple experimentation suggests that a value of Z=0.3 is suitable for the videos tested.

The threshold Z must be high enough to prevent non-butterflies from being matched and yet low enough to ensure that all butterflies are correctly identified. To reduce the number of unmatched butterflies, a spatially aware butterfly buffer was created. For all identified butterflies, the buffer stores the butterfly class, location in the frame and matching strength for the previous 30 frames (1 second). If a foreground object is found but the matching strength is below the threshold, the buffer is checked to determine if a butterfly of the same class has been found in a nearby location of the frame 10 or more times in the last 30 frames. If so, it is deemed to match that butterfly class despite the match strength being below the threshold value.

B. *Implementation*

We implemented our algorithm in C++, utilising both the OpenCV and BazAR computer vision libraries. C++ was chosen to maximise runtime efficiency to allow for future realtime processing.

The trees were trained with 11 different butterfly classes; four training images per class. The species we used are: *Euthalia monina, Lexias pardalis, Pachliopta aristolochiae, Precis atlites, Precis iphita, Rhinopalpa polynice, Symbrenthia hypatia,Terinos atlita, Thauria aliris, Trogonoptera brookiana, Troides amphrysus*.

Figure 3. Sample training images for Thauria aliris

The training images (Fig.3) were taken with a Canon 550D in auto mode at maximum resolution, later resized to approximately twice the size of the actual butterflies in the video. The background was set to white for the training images which were saved in the common JPEG format (.jpg). The average image size was approximately 200x200 pixels.

We experimented with 14 different video samples, all with an initial background learning phase in which no butterflies were present. The learning phase was between 80-200 frames of video depending on light changes, camera movement and tree/leaf movement in the frame. The video was recorded at 640x480 resolution at a frame rate of 30 frames per second with a Sony TX5 Cybershot digital camera. The videos varied widely in terms of the butterflies present in the frame. The videos all contained 1-3 butterflies at any time after background learning and were saved in the QuickTime file format (.mov).

Training images and videos were recorded at the Kuala Lumpur Lake Gardens Butterfly Park over multiple days. A GorillaPod style tripod was used to secure the camera recording the footage to a tree that was near to butterfly feeding platforms to ensure a large volume of butterflies could be filmed.

The training phase of the program was completed in approximately 13 seconds when run on a Dell Studio XPS with an Intel Core i7 Q720 (1.6GHz, Turbo Boost to 2.8GHz).

V. RESULTS

The results obtained between consecutive runs of the program varied due to the random tests in the internal nodes of the trees and the random transforms generated for the training images. The random number generators were seeded with the system time to ensure variation. We experimented with 13 separate video clips that were cut from large blocks of video (to eliminate periods with no butterflies in the frame). A small number of clips were rejected due to violent camera movement when the tripod slipped or when the sunlight changed abruptly with butterflies in the frame as dynamic background re-learning works only when no butterflies are present.

The different videos yielded different matching results between consecutive runs due to the random nature of the transforms and tree tests. Figure 4. summarises the matching performance rates for each video for three separate runs of the program. Table III details results and variation over three trial runs.

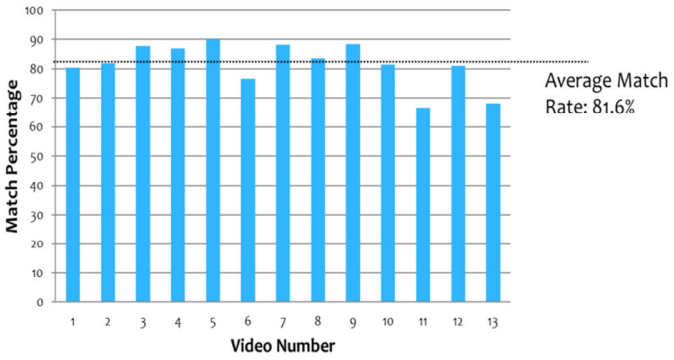

Figure 4. Average results of 13 video footages

VI. CONCLUSION

We have successfully developed an automated butterfly species identification system that is capable of identifying resting and flying butterflies in real-time in their natural environment. The system has achieved an average recognition rate of 82% over 10 butterfly species.

The use of random trees to identify butterflies was found to be quite successful. Random trees allowed for fast runtime

performance by shifting computational load to the offline training phase of the program in which randomly transformed images were used to create probability distributions in the tree leaf nodes. The tree tests we developed produced better than expected results. The tests were simple and computationally inexpensive and we believe there is still scope to improve accuracy.

The methods employed in our system have great potential for future applications in the field. It is estimated that matching rates could be increased to greater than 95% whilst running at or above real-time with further improvements.

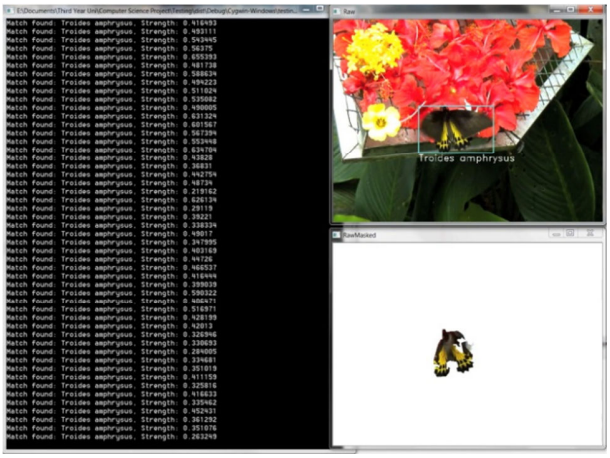

Figure 5. Sample screen capture of program. Note the box drawn over the identified butterfly. The bottom right shows the background subtraction.

REFERENCES

[1] Summerville K, Ritter L, Crist T, "Forest moth taxa as indicators of lepidopteran richness and habitat disturbance: a preliminary assessment", *Biological Conservation*, 2004:116, 9-18

[2] Loke K S, Egerton S, "Automated Eye on Nature (AEON) and the Were-Tigers of Belum", Proc. International Conference on Intelligent Environment 2010, Malaysia, Aug. 2010.

[3] Yun S K, "Hierarchical Recognition: Taxonomy of Animals", Computer Science and Artificial Intelligence Laboratory, MIT, Cambridge, MA, USA, 2007

[4] Kastberger G, Radloff S, Kranner G, "Individuality of wing patterning in Giant honey bees (*Apis laboriosa*)", *Apidologie,* 2003:34, 311-318

[5] Tilo Burghardt, Janko Calic, Analysing Animal Behaviour in Wildlife Videos Using Face Detection and Tracking. IEEE Proceedings - Vision, Image, and Signal Processing, 153(3). ISSN 1359-7108, pp. 305–312. June 2006.

[6] P. Viola and M. Jones, "Robust Real-time Object Detection," Second International Workshop on Statistical and Computational Theories of Vision, Vancouver, USA, 2001.

[7] Tilo Burghardt, Richard B Sherley, Peter J Barham, Non-invasive Monitoring of Colonial Seabirds using Computer Vision: Scope and Limitations as Exemplified on African Penguins and Bank Cormorants. 1st World Seabird Conference (WSC2010), pp. C4-1–11 ff.. September 2010.

[8] Richard B Sherley, Tilo Burghardt, Peter J Barham, Innes C Cuthill, Computer Vision Monitoring of African Penguins Spheniscus demersus: New Data on the Field Capacity from Robben Island. 7th International Penguin Conference (IPC2010), pp. 116 ff–. August 2010.

[9] Júlia Reisser, Maíra Proietti, Paul Kinas and Ivan Sazima, "Photographic identification of sea turtles: method description and validation, with an estimation of tag loss," Endangered Species Research Vol. 5:73-82, 2008, doi:10.3354/esr00113

[10] Michael Mayo and Anna T. Watson, "Automatic species identification of live moths," Knowledge-Based Systems, Volume 20 Issue 2, March, 2007, doi: 10.1016/j.knosys.2006.11.012.

[11] A. Watson, M. O'Neill, I. Kitching, "Automated identification of live moths (Macrolepidoptera) using Digital Automated Identification System (DAISY)," Systematics and Biodiversity 1 (3) (2004) 287-300.

[12] V. Lepetit and P. Fua. Keypoint recognition using randomized trees. IEEE Transactions on Pattern Analysis and Machine Intelligence, 28(9):1465–1479, September 2006.

[13] Butler D, Bove Jr. V, Sridharan S, 'Real-Time Adaptive Foreground/Background Segmentation', EURASIP Journal on Applied Signal Processing, 2005:14, 2292–2304.

[14] Kim K, Chalidabhongse TH, Harwood D, Davis L, 'Real-time foreground–background segmentation using codebook model', Real-Time Imaging, 2005:11, 167–256.

TABLE III. DETAILS OF RESULTS OF THREE RUNS, VARIATION DUE TO RANDOM CHOICE POINTS

Video Filename	Video Length (frames)	Average Match Rate for Video (%)		
		Run 1	Run 2	Run 3
1.mov	232	81.35593	81.22270	78.66108
2.mov	988	82.06478	81.68016	81.47773
4.mov	999	89.08909	87.28728	86.38639
5.mov	280	96.78571	83.57143	80.35714
6.mov	557	89.58707	91.02334	89.58707
7.mov	1178	76.14595	79.62806	74.04611
8.mov	1710	90.05847	84.50292	89.70760
9.mov	719	82.89291	84.28373	83.03199
10.mov	1056	89.96212	87.97348	86.70455
11.mov	164	91.46341	67.07317	85.71429
12.mov	89	67.41573	64.04494	68.53933
13.mov	224	83.03571	79.46428	80.35714
14.mov	139	65.46762	61.15107	77.69784
Overall Average Matching Rate (%)		81.55126		