Insect inspired view based navigation exploiting temporal information [★]

Efstathios Kagioulis^{1[0000–0001–8088–3904]}, Andrew Philippides^{1[0000–0001–5503–0467]}, Paul Graham^{1[0000–0002–3745–0940]}, James C. Knight^{1[0000–0003–0577–0074]}, and Thomas Nowotny^{1[0000–0002–4451–915X]}

Centre for Computational Neuroscience and Robotics, University of Sussex, Brighton BN1 9QJ, UK e.kagioulis@sussex.ac.uk

Abstract. Visual navigation is a key capability for robots. There is a family of insect-inspired algorithms that use panoramic images encountered during a training route to derive directional information from regions around the training route and thus subsequently visually navigate. As these algorithms do not incorporate information about the temporal order of training images, we describe one way this could be done to highlight this information’s utility. We benchmark our algorithms in a simulation of a real world environment and show that incorporating temporal information improves performance and reduces algorithmic complexity.

Keywords: Visual navigation · Visual homing · Bio-inspired robotics

1 Introduction

Foraging ants have impressive visual navigation capabilities [26]. They can use celestial visual information as a compass for Path Integration (PI) [19] and use learnt visual memories to guide complex routes [15] between their nest and a food source. The ability to travel long routes, guided by stored visual scenes, prompts the question of how the visual memories are organised in the small brains of insects [2].

We and others have developed a series of algorithms mimicking ant route navigation in which ants use a set of views or *snapshots* experienced during a first (PI-mediated) training route to later navigate by rotating and comparing the rotated current view with the stored training route views [22, 2, 13]. Because the route memories are stored when travelling facing forwards, adopting a heading which is familiar to them implicitly means that the agent is likely near the route and facing in a similar direction as when last near that location. In this way, finding familiar views means recovering the correct direction to move in and thus routes can be successfully navigated. While these algorithms vary in the way that the route memories are stored, being either used to train an artificial neural network (ANN) to learn a compact encoding of the route information [22,

[★] This work was supported by EPSRC grants EP/P006094/1 and EP/S030964/1 and a University of Sussex doctoral scholarship

18], or instead storing all memories exactly as experienced, (the so-called perfect memory, henceforth, PM, algorithm [3]) what they have in common is that all views are used for the comparison and the sequence they are encountered in is disregarded. For instance, in PM at each step, agents compare the current views to *all* stored snapshots without any consideration of their order.

This means that, while they can quite robustly navigate a variety of environments, they can fail in places where images are very similar (e.g. where a route crosses itself or is very tortuous with different movement directions in close proximity [12]). In such cases it is not obvious which part of the stored route should be used to provide the correct movement direction. In addition, throwing away sequence information and storing/using all the memories to generate a heading is highly inefficient. For instance, for PM, both memory and computation scale linearly with route length, leading to it being deemed computationally impractical and biologically implausible [3] despite generally having the most accurate performance [2].

Given these limitations, it is reasonable to think of alternative ways that stored views could be organised. One possibility is that views could be stored as a sequence, and whilst this could be problematic [25, 21], sequence information has been shown to be useful in engineering approaches to navigation [17, 5]. Sequence has also been considered as a solution to image distortion due to camera/head tilt [1]. Furthermore, there are some suggestions that ants also use some notion of sequence when recapitulating a route using visual memories [8, 23].

In this paper we thus investigate how incorporating the temporal sequence in which snapshots are experienced can improve both run-time and accuracy of the aforementioned route navigation algorithms. To enable us to clearly assess the benefits and weaknesses of this approach, we use temporal information in a purposefully simple way to augment the PM algorithm so that the agent considers only a small sub-sequence of the route memories to extract direction, rather than all of them.

2 Methods

To evaluate the efficacy of our algorithm, we use a virtual environment modelled on the environment of desert ants [13] (Fig 1 a,b for top down views). We recover panoramic views from a perspective similar to that of an ant (Fig. 2 for example views). As our algorithm relies on comparing stored snapshots and current views, we first describe the image comparison functions, before outlining our new Sequential Memory Window (SMW) algorithm and describing the image pre-processing used. All code was written in python 3.5 using NumPy [20] OpenCV [4] and pandas [16] libraries.

Image comparison functions: We use two methods for quantifying the difference between a current view and a stored snapshot. First, we use the Root Mean Squared Error image difference function (IDF) [28]:

$$IDF(C(\mathbf{x}, \theta), S(\mathbf{y}, \phi)) = \sqrt{\frac{\sum_{i=1}^M \sum_{j=1}^N (C_{ij}(\mathbf{x}, \theta) - S_{ij}(\mathbf{y}, \phi))^2}{M \times N}} \quad (1)$$

where $C(\mathbf{x}, \theta)$ is an $M \times N$ panoramic view from position $\mathbf{x}$ at an orientation θ with pixel greyscale values C_{ij} at row i and column j and $S(\mathbf{y}, \phi)$ is a view stored from position $\mathbf{y}$ at orientation ϕ with pixel values S_{ij} .

The second method uses the correlation coefficient (CC) to derive the similarity between images [24] via:

$$CC(C(\mathbf{x}, \theta), S(\mathbf{y}, \phi)) = \frac{\sum_{i=1}^M \sum_{j=1}^N (C_{ij} - \bar{C})(S_{ij} - \bar{S})}{\sqrt{\sum_{i=1}^M \sum_{j=1}^N (C_{ij} - \bar{C})^2} \sqrt{\sum_{i=1}^M \sum_{j=1}^N (S_{ij} - \bar{S})^2}} \quad (2)$$

where $\bar{C}$ and $\bar{S}$ are the mean pixel values of $C(\mathbf{x}, \theta)$ and $S(\mathbf{y}, \phi)$ respectively.

To derive a heading from comparing $C(\mathbf{x}, \theta)$ and $S(\mathbf{y}, \phi)$, we ‘rotate’ the current view by varying θ through 360 degrees (in steps of one degree) and use either the IDF or CC to determine the orientation $\hat{\theta}$ at which C best matches S . For the IDF a smaller value indicates a higher similarity while for CC it is the highest value. In both cases the agent will move in the direction that is most familiar thus *pairing visual similarity to action*.

Route navigation using a sequential memory window: In our route navigation algorithms, the agent first traverses a predefined route (e.g. blue line in Fig 1 (A)) during which it periodically stores views, or snapshots. In the experiments in this paper, snapshots are stored every 1 cm and the facing direction of each snapshot is set as the direction to the next snapshot (or the direction of the penultimate snapshot in the case of the last snapshot in the sequence). Unlike our previous work (e.g. [3]), the sequence of the snapshots during the training route is retained, although the distance between them (and as a corollary, the fact that they are regularly sampled) is not important. The agent subsequently navigates by ‘rotating’ on the spot and comparing rotated versions of the current view individually to a subset of the stored snapshots using either the IDF or CC as described in the previous section. The direction which results in the lowest minimum for the IDF/highest maximum for the CC across the subset of snapshots is selected as the movement direction.

In contrast to the standard PM model [3] which compares the current view to all snapshots, the sequential memory window model compares the current view only to a subset of n consecutive images of the image memory. The best match is used to determine both current movement direction and the start position of the memory window for the next comparison. While other options are possible for initialising the next window, this is reasonable under the assumption that the agent moves fairly directly from start to end and that test points are well-spaced. For the first window, as the test positions are near the start of the route, we use the first snapshot (see Discussion for alternatives).

Pre-processing: All panoramic images are rendered at an original resolution of $[720 \times 150 \times 3]$ in RGB. We typically transform the image to a $[360 \times 75]$ greyscale image using openCV’s resize function using bilinear interpolation. As it has been shown that resolution of the images can influence homing accuracy [27, 17] and we know that ants have low resolution vision $>1^\circ$ per pixel and use

light intensity rather than colour to navigate [9] we transform the image to greyscale and resize it to what, for an ant, would be high-resolution $[360 \times 75]$ (1 degree per pixel), medium resolution $[180 \times 50]$ (2 degrees per pixel) and low resolution $[90 \times 25]$ (4 degrees per pixel). For pre-processing we compared edge detection and blurring. For blurring, we use a 5×5 Gaussian kernel (low pass filter) with mean 0 and standard deviation 1 in order to remove high-frequency noise and smooth out the image. As the kernel is always the same number of pixels, when we resize the image we effectively scale the amount of smoothing. Fig 1 c) shows a panoramic image from AntWorld after resizing and blurring. There is some experimental evidence [7] that some ants navigate using simple cues on the horizon as a well defined boundary to derive directional information. As edge detection can separate sky, tussocks and ground, we also test Canny edge detection with a lower bound of 180 and upper bound of 200 (pixel values scale [0-255]) [10].

3 Results

To test the algorithms, we derive headings from a grid of test positions spaced in 10cm increments around example routes. All variants are tested against 10 different routes and errors are calculated as the absolute difference between the derived heading and the heading of the nearest snapshot.

Temporal windows improve visual aliasing problems: To illustrate the benefits of our sequence based algorithm, we first compare results on a route with a loop in it. The agent is given a training route (blue lines in Fig 1) and we derive the headings that the agent would follow by each algorithm for all points within 10 cm of the route. The derived headings are indicated in Fig 1 by blue arrows for the ones that are close to correct and by red arrows for large errors (> 40 degrees). We highlight errors > 40 degrees only to demonstrate the behaviour of SMW against standard PM for a given loop section of a route. In many robotics tasks this error is not acceptable but here we are most concerned with showcasing the difference between algorithms and so highlight large deviations from the route headings.

Errors in orientation when using PM occur because of visual aliasing where a view will match with a route memory that was stored in a completely different location [12]. In other words the current view matches with a memory that looks familiar but is spatially different thus sending the agent in the wrong direction. There are multiple interlinked reasons for such aliasing. It could be that the view from two route locations is very similar. To illustrate the problem we look at a route with a loop.

The first point where we see differences in performance is near the loop intersection Fig. 1. Errors near the loop intersection are due to aliasing where perfect memory matches current views to memory views that are captured either at a different point in the route or near the route intersection in the opposite direction. When the agent is at the intersection, there are two ‘correct’ directions: either continuing along the route or to omit traversing the loop and turn at

Fig. 1: Two traversals of the same testing route (blue solid line) that contains a loop. Red arrows are errors above a threshold of 40° . **A** Traversal using the standard algorithm. **B** Traversal using sequential perfect memory.

the loop intersection. In the PM version we see that the agent chooses which direction based on proximity to the respective part of the route, assuming that no occlusions are affecting the matching. In contrast, because the sequence only searches the images close to the previous match, it gets the correct direction by entering the loop without considering the memories from the perpendicular direction. The same goes for the case of exiting the loop. Of course this is a trivial example, if insects had looped routes, they could deploy other measures, (by reducing the search angle for example to $\pm 90^\circ$) although it highlights the key point that temporal information can be useful.

However, if there is partial occlusion from dense vegetation or other obstacles, the sequence can resolve the problem. In these cases, the occluded part of the view often matches noisily while the clear view part can match with partial features from other points in the route. Due to the smaller number of features, there are more aliased points than there would be normally. In this scenario, the window means that the agent is constrained to match the current view with a memory that is spatially nearby and free from occlusions. This means that the partial features it *does* match are likely to be the correct ones, thus keeping the agent on track without generating a lot of angular error. Fig 2 demonstrates how an occluded window image Fig2(B) can disrupt image matching with a non-occluded current view 2(A). However, because stored memories within the window Fig2(C) that are free from occlusion provide reasonably similar features they can provide a match that is more likely to keep the agent in the correct direction than images further afar on the route might.

Window size: The size of the window appears to have a large effect on the degree of error. For very small window sizes, the algorithm produces significant

Fig. 2: A situation in which the SMW can be beneficial. In this case the closest image to the test image (**A**) in space was **B** (window image 47) but, because of occlusion the images do not match. However, because of the SMW, the algorithm matched with **C** (window image 39) 8 index places behind even though it also did not match particularly well, instead of matching with an almost random match from the entire route. As a result the agent typically will make less severe errors due to occlusion and/or aliasing.

angular error which can be attributed to the window falling behind the actual position of the agent. A smaller window also provides less chances for the agent to match a clear route image when the window contains occluded images. For example, if the window at a given point is part of a route section where views are occluded the matching is going to be false (both in term of the image memory and orientation). On the other hand, if the window is larger, the agent has more chances to match with a memory that is further ahead and not occluded – thus producing a good match.

The error decreases as the window size increases and becomes stable at an average of 7.1° for window sizes between 13 and 16 images (Fig 3). The error then increases again and, for window sizes greater than 18, it is comparable to Full Perfect Memory at a average of 12.8° . Full perfect memory (PM) can also be thought of as an SMW with a window the size of the entire memory. Within the appropriate range of window sizes, SMW produces half the mean error and a flatter distribution of error than PM. The lower mean error can be attributed to

Fig. 3: Average error along routes for different window sizes. Images are compared using IDF matching and the lowest resolution (4° per pixel).

correlation coefficient being able to deal with occlusions and generate a smoother familiarity gradient compared to IDF.

Image matching: Correlation coefficient matching is superior to IDF matching. For smaller window sizes, both methods produce large errors for the reasons mentioned in the previous section. When pre-processing with blur, the IDF requires a larger window size to converge to the same low level of error as CC. After window size 13, the error resulting from both approaches is equivalent. Combining IDF with edge pre-processing yields high error signals even when the window size is large which we believe is due to occlusion having more drastic effects on edges than it does on pixels. Our analysis showed that, when the memory window contains an almost fully occluded image, test images tend to strongly match with it. This is because when there are very few edges, the IDF is small even if the edges do not match at all. The match is so strong that the window stays trapped in that area of the memory for the remainder of the route. As the data show (Fig5) the IDF performs relatively well with edge pre-processing, this issue can potentially be avoided by increasing the window size.

In contrast, correlation matching performs better when combined with edge detection than it does with blur. This is because a lot of noise is removed by the edge detector. In addition, the correlation is normalised by the standard deviation of the images, so that it measures better how edges match regardless of how many edges there are. This avoids the false matches with occluded (edge-poor) views we discussed above for the case of the IDF.

Pre-processing: Pre-processing with blur and correlation matching combined with window sizes between 11 and 15 produce very similar errors independent of the resolution, allowing us to use lower resolution and reduce computation (Fig 5a). The distributions of observed errors when using high resolution edges and CC matching is lower than the best observed error for blur pre-processing (Fig 5b). The errors are smaller in this condition but also, for win-

Fig. 4: Split violin plot showing the error distribution for correlation and IDF matching with different window sizes. **A** With blurred images, correlation matching converges to a minimum faster than IDF matching but window sizes 13 to 17 both exhibit similar behaviour. **B** With edge detection pre-processing, the IDF never converges to acceptable error levels while correlation matching converges from window size 10 onwards.

Window sizes between 10 and 20, the error distribution does not appear to change, being consistently lower than the best blur based scores. In this case high-res edges consistently outperform low and mid-res edges. This could be accredited to edge detection removing a lot of noise from the images and additionally high-res edges are more detailed thus allowing for more precise matching. Occlusions do not contribute as much to the matching as their edges generate less differences between the images compared to the sum of the pixels in the fill of the objects.

Time complexity: Time complexity between the full perfect memory and sequential perfect memory is significant. Perfect memory compares every test image with all route images for all angles (assuming a 0 to 360 scan). Sequential

Fig. 5: Each line represents the 90th percentile of the error distribution for correlation matching with various resolutions for small to large window sizes. a) Images are blurred and down-sampled to high, low and medium resolution. b) Edge detection applied paired with high, low and medium resolution.

perfect memory compares each test image to images within a window – a small fraction of the route memory. For all 10 routes tested, there a little over 800 route images and the best performing window size is 17 images long. Therefore, the perfect memory time complexity is $O(N * R * \theta)$ where N is the number of test images, R the number of route images and θ the degrees of search (360). For the sequential perfect memory, the time complexity is $O(N * W * \theta)$ where N and θ are the same as above and W is the width of the window which can range between sizes 10 and 20 and, by definition, is much smaller than the entire route memory (Fig. 6). It is worth mentioning that the runtime of the full perfect memory is bound to grow with the size of the route while the sequential version is affected only by the size of the window (This is true ONLY for the static database tests as in a real world scenario we would have to consider the speed of the agent as well. But then again with respect to each test position sequential perfect memory would be faster).

4 Discussion

In this paper we have shown that we can both increase the accuracy of navigation, and reduce the computational cost, by including temporal sequence information in the stored route from training. By focusing on the areas where the algorithm fails which are typically caused by partial occlusion due to vegetation, we showed that performance could be further improved by using correlation-based image comparisons as well as edge enhancement. While changing resolution had a relatively minor effect in terms of improving the best combination

Fig. 6: a) Runtime for PM (y axis) versus SMW (x axis). For the timing we used medium resolution (180, 50) and blur, IDF and the window size 17. SMW is about 10 times faster than PM. b) Runtime and mean error for SMW using IDF or CC. Point represents data for window sizes from 6 through to 20. IDF is much faster but generates larger errors. For CC there is an optimal compromise for the error versus runtime trade-off at window size around 10.

of window length and processing parameters, it is clear that there will be an optimal resolution for a given environment, at the cost of increased algorithmic complexity. Likewise, finding the optimal window size is likely to be environment specific and, given that in some situations, accuracy gains are marginal, it might be that the choice is determined by computing time.

For sequence based navigation models, the length of the comparison window is important and not all memory window lengths work. Indeed, we saw that windows that are too small can fail fairly spectacularly. Examining this issue reveals the weakness of this method. If the algorithm gets ‘lost’ due to an erroneous match, the window can get stuck at that point and never find the correct matching point for subsequent test points. The smaller the window size and the more cluttered the environment, then the higher chance of the window getting ‘stuck’ and sending the agent in the wrong direction. This is linked to the issue of at what point to start the algorithm and both points could be solved in a similar way. That is, if the starting point is unknown, or the best match is deemed poor, the algorithm needs to run a full memory scan in order to set the window. However, this uncertainty of “where in the world am I?” along with “which of my memories are useful now?” is the fundamental problem of any algorithm of this nature, as by restricting matches one is, amongst other things, constraining the information available to the algorithm. Thus, while a good starting position and initial match could result in a good run for the agent, the opposite can be true. Another limitation is in nest search, where the agent must precisely

home from all locations, rather than general route following. To ensure they can find the nest, ants perform learning walks immediately after leaving it, in which snapshots facing the the nest from multiple locations are stored. In this case the sequence information will not be important/useful as these memories should all be used to find the nest. At that point a mechanism is required where the agent switches off sequence information and compares across all memories.

In the work presented here, we have used a simple way of using sequence information, aimed at showing the utility of incorporating temporal information in a route navigation algorithm. Other methods of using sequence information may ameliorate the aforementioned issues. We could for instance, update the window such that the best previous match is at the middle or implement a dynamic window length which would grow if matches were deemed poor [11]. However, it is perhaps more profitable to consider complex matching processes and ways of incorporating temporal information. For instance SeqSLAM [17] compares multiple memories within a window to multiple current views and can thus be robust to some bad matches. Another option would be to use artificial neural networks that encode temporal information [6, 14] and use these to learn a combined representation of sequence, or indeed position, and route snapshots as in [22]. It is unclear how one would implement a neural network to encode both image snapshots as well as sequence information, but incorporating PI information might be possible, and is the subject of future work.

Our priorities in developing these models is to push the algorithms for use in closed-loop, real world, natural environments as experienced by ants. Assessing how temporal information is useful in such environments, and comparing it to what we know about how temporal information is used by insects, could allow us to not only improve robotic navigation but also illuminate insect neurobiology.

References

1. Ardin, P., Mangan, M., Wystrach, A., Webb, B.: How variation in head pitch could affect image matching algorithms for ant navigation. *J Comp Physiol A* **201**(6), 585–597 (2015)
2. Ardin, P., Peng, F., Mangan, M., Lagogiannis, K., Webb, B.: Using an insect mushroom body circuit to encode route memory in complex natural environments. *PLoS Comput Biol* **12**(2) (2016)
3. Baddeley, B., Graham, P., Husbands, P., Philippides, A.: A model of ant route navigation driven by scene familiarity. *PLoS Comp Biol* **8**(1) (2012)
4. Bradski, G.: The OpenCV Library. Dr. Dobb’s Journal of Software Tools (2000)
5. Chancán, M., Hernandez-Nunez, L., Narendra, A., Barron, A.B., Milford, M.: A hybrid compact neural architecture for visual place recognition. *IEEE Robotics and Automation Letters* **5**(2), 993–1000 (2020)
6. Funahashi, K.i., Nakamura, Y.: Approximation of dynamical systems by continuous time recurrent neural networks. *Neural networks* **6**(6), 801–806 (1993)
7. Graham, P., Cheng, K.: Ants use the panoramic skyline as a visual cue during navigation. *Curr Biol* **19**(20), R935–R937 (2009)
8. Graham, P., Mangan, M.: Insect navigation: do ants live in the now? *J Exp Biol* **218**(6), 819–823 (2015)

9. Graham, P., Philippides, A.: Vision for navigation: What can we learn from ants? *Arthropod structure & development* **46**(5), 718–722 (2017)
10. Green, B.: Canny edge detection tutorial. Retrieved: March 6, 2005 (2002)
11. Hausler, S., Jacobson, A., Milford, M.: Multi-process fusion: Visual place recognition using multiple image processing methods. *IEEE Robotics and Automation Letters* **4**(2), 1924–1931 (2019)
12. Knight, J.C., Sakhapov, D., Domcsek, N., Dewar, A.D., Graham, P., Nowotny, T., Philippides, A.: Insect-inspired visual navigation on-board an autonomous robot: real-world routes encoded in a single layer network. In: *The 2018 Conference on Artificial Life: A Hybrid of the European Conference on Artificial Life (ECAL) and the International Conference on the Synthesis and Simulation of Living Systems (ALIFE)*. pp. 60–67. MIT Press (2019)
13. Kodzhabashev, A., Mangan, M.: Route following without scanning. In: *Conference on Biomimetic and Biohybrid Systems*. pp. 199–210. Springer (2015)
14. Lin, T., Horne, B.G., Tino, P., Giles, C.L.: Learning long-term dependencies in naxr recurrent neural networks. *IEEE Transactions on Neural Networks* **7**(6), 1329–1338 (1996)
15. Mangan, M., Webb, B.: Spontaneous formation of multiple routes in individual desert ants (*cataglyphis velox*). *Behav Ecol* **23**(5), 944–954 (2012)
16. Wes McKinney: Data Structures for Statistical Computing in Python. In: Stéfan van der Walt, Jarrod Millman (eds.) *Proceedings of the 9th Python in Science Conference*. pp. 56 – 61 (2010). <https://doi.org/10.25080/Majora-92bf1922-00a>
17. Milford, M.J., Wyeth, G.F.: Seqslam: Visual route-based navigation for sunny summer days and stormy winter nights. In: *2012 IEEE International Conference on Robotics and Automation*. pp. 1643–1649. IEEE (2012)
18. Müller, J., Nawrot, M., Menzel, R., Landgraf, T.: A neural network model for familiarity and context learning during honeybee foraging flights. *Biol Cybern* **112**(1-2), 113–126 (2018)
19. Müller, M., Wehner, R.: Path integration provides a scaffold for landmark learning in desert ants. *Curr Biol* **20**(15), 1368–1371 (2010)
20. Oliphant, T.: NumPy: A guide to NumPy. USA: Trelgol Publishing (2006–), <http://www.numpy.org/>, [Online; accessed jtoday]
21. Philippides, A., Baddeley, B., Cheng, K., Graham, P.: How might ants use panoramic views for route navigation? *J Exp Biol* **214**(3), 445–451 (2011)
22. Philippides, A., Graham, P., Baddeley, B., Husbands, P.: Using neural networks to understand the information that guides behavior: a case study in visual navigation. In: *Artificial Neural Networks*, pp. 227–244. Springer (2015)
23. Schwarz, S., Mangan, M., Webb, B., Wystrach, A.: Route-following ants respond to alterations of the view sequence. *J Exp Biol* (2020)
24. Sedgwick, P.: Pearson’s correlation coefficient. *BMJ* **345**, e4483 (2012)
25. Smith, L., Philippides, A., Graham, P., Baddeley, B., Husbands, P.: Linked local navigation for visual route guidance. *Adaptive Behavior* **15**(3), 257–271 (2007)
26. Wehner, R., Boyer, M., Loertscher, F., Sommer, S., Menzi, U.: Ant navigation: one-way routes rather than maps. *Curr Biol* **16**(1), 75–79 (2006)
27. Wystrach, A., Dewar, A., Philippides, A., Graham, P.: How do field of view and resolution affect the information content of panoramic scenes for visual navigation? a computational investigation. *J Comp Physiol A* **202**(2), 87–95 (2016)
28. Zeil, J., Hofmann, M.I., Chahl, J.S.: Catchment areas of panoramic snapshots in outdoor scenes. *JOSA A* **20**(3), 450–469 (2003)