Dual Population Coding for Path Planning in Graphs with Overlapping Place Representations

Hanspeter A. Mallot[1][0000−0003−4208−4348], Gerrit A. Ecke[1], and Tristan Baumann[1]

University of Tübingen, Germany
{hanspeter.mallot,gerrit.ecke,tristan.baumann}@uni-tuebingen.de

Abstract. Topological schemes for navigation from visual snapshots have been based on graphs of panoramic images and action links allowing the transition from one snapshot point to the next; see, for example, Cartwright and Collett [5] or Franz et al. [9]. These algorithms can only work if at each step a unique snapshot is recognized to which a motion decision is associated. Here, we present a population coding approach in which place is encoded by a population of overlapping "firing fields", each of which is activated by the recognition of an unspecific "micro-snapshot" (i.e. feature), and associated to a subsequent action. Agent motion is then computed by a voting scheme over all activated snapshot-to-action associations. The algorithm was tested in a large virtual environment (Virtual Tübingen [24]) and shows biologically plausible navigational abilities.

Keywords: Topological navigation · State-action graph · Population coding

1 Introduction

1.1 Parsimonious representations of space

The evolution of spatial cognition in animals started from simple stimulus-response behaviors such as stimulus-driven orienting reactions, and proceeded further by a number of innovations that include (i) mechanisms for egomotion perception and path integration, (ii) the memorization of stimulus-response pairs composed of a distinguishable landmark and a navigational action ("recognition-triggered response"), (iii) the concatenation of such recognition-triggered responses into chains or routes, and (iv) the linking-up of multiple recognition-triggered responses into networks or graphs in which novel routes can be inferred by the combination of known route segments [23, 13, 26, 12]. In addition, mechanisms for invariant landmark and place recognition, strategic selection of way- or anchor-points, metric embedding of place-graphs, or hierarchical graph structures may improve navigational performance and are thus likely to play a role.

Since many different models can be build on these elements, it is of interest to ask for a minimal or most parsimonious model supporting a given level of

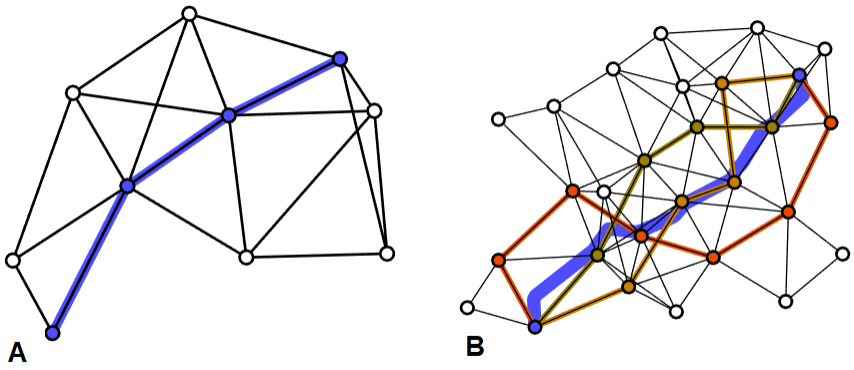

Fig. 1. Graph-navigation in single-unit and population coding. A: In standard topological navigation, every place is represented by a unique node and route segments are given by the graph-links. The desired trajectory shown in blue is therefore a graph path. B: In dual population coding, a bundle of paths is constructed for a given navigation problem. Three such paths without common nodes (except start and goal) are shown in red, orange, and yellow colors in the figure. The desired trajectory (shown in blue) is then calculated by a voting scheme over the currently visible nodes of all paths. It is generally not a path of the graph.

behavioral flexibility. In this paper, we address this question for the case of the minimal cognitive architecture supporting way-finding behavior. By a minimal model, we mean a model meeting the following requirements:

1. A minimal model should be close to the evolutionary starting point of stimulus-response schemata;
2. it should require only a small amount of visual invariance in object recognition and therefore work with the rawest possible image information;
3. it should use simple decision processes in path planning such as recognition-triggered responses; and
4. it should not rely on explicit metric information which is hard to obtain.

With these constraints in mind, we present a model for graph-based navigation that marks a lower bound of cognitive complexity required for way-finding and that can be used to study further improvements resulting from additional evolutionary innovations.

The model presented in this paper is not primarily about the hippocampal system for place as is known from rodents and some other mammalian groups, although some inspiration has been drawn from these results. Our main interest, however, is a *computational theory of navigation* based on devices such as snapshots and state-action schemata. Such computational theory will have implications for navigational behavior in insects, mammals, humans, and even robots.

1.2 Dual population coding

Two basic elements of spatial representations are (i) stimulus-response schemata such as Tolman's [22] means-ends-relations, O'Keefe and Nadel's [17] taxon system, Kuipers' [10] control laws, or the place-recognition-triggered response of Trullier et al. [23], and (ii) the representations of places and place relations such as Cartwright and Collett's [4] snapshot-codes for places or O'Keefe and Nadel's [17] locale system. The two systems are connected by the role that place recognition takes as a "stimulus" in the stimulus-response schemata involved.

In the classical state-action-approach, it is assumed that each place is a state represented by just one node of a graph [10, 16, 9, 11] such that a unique state-action schema will control each navigational step. If place recognition fails, navigation will go wrong. Robustness of navigation therefore depends foremost on the robustness and invariance of place recognition as a prerequisite. Here, we argue that in an evolutionary view of navigation, robust pathfinding should be possible even with rudimentary place recognition and distributed place representations.

Our model differs from standard models of topological navigation [10, 16, 9, 11] in two major respects that can be summarized as "dual population coding" (see Figure 1): first, at any instant in time, many nodes of the graph are activated and encode the agent's position in a population scheme. This avoids costly selection processes of strategic anchor points and has the additional advantage that the visual cues and recognition processes can be kept simple. Of course, population coding of space is well in line with empirical findings in the place-cell literature [27]. Second, as a consequence of population coding of space, route selection has to be based on many interacting recognition-triggered response schemata, one for each active unit in the population code. This is implemented by a voting scheme where the suggested motion decisions from all active schemata are averaged. The idea of view voting has been suggested earlier for human behavioral data [14]. In insect navigation, a similar scheme has been suggested for route following with multiple snapshots by [20, 1, 7], but unlike our model, these models do not allow for alternative route decisions from a given position. As a result of dual population coding, the trajectory eventually found by the algorithm is not a path of the graph, but a metric average of bundles of many paths connecting individual nodes in the population codes for start and goal.

2 Navigation algorithm

This section will explain the algorithm in five steps, starting from the initial definition and later matching of features, and proceeding to the learning of graph edges and their directional labels. Once the graph is learned, the voting scheme is applied for pathfinding. All examples shown are taken from a virtual reality implementation where a simulated agent is exploring and later navigating in the "Virtual Tübingen" environment [24].

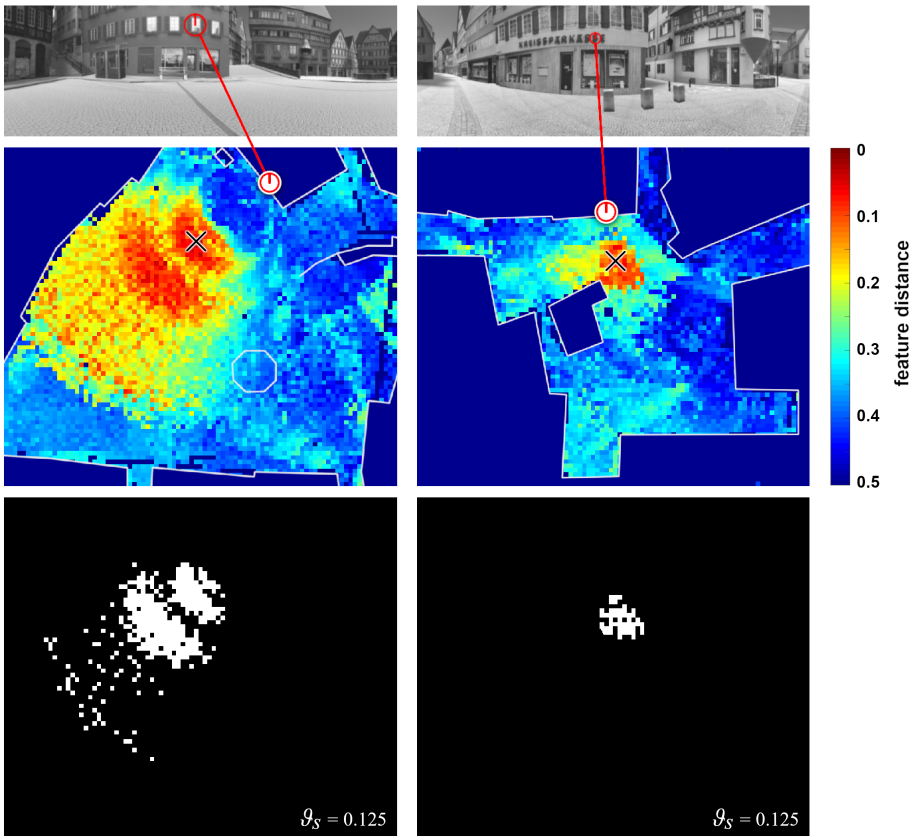

Fig. 2. Place fields. Top row: two views from a scene with detected features (a window and a letter from a company nameplate). Middle row: local maps of the environment superimposed with the similarity of the reference feature to any feature detected from each position in the map ($\min_i \|\boldsymbol{d}_{ref} - \boldsymbol{d}_{i,\boldsymbol{x}}\|^2$). The black $\times$ marks the position where the reference feature was first detected. Third row: same map with points where one feature was identified with the reference feature, based on both the similarity criterion and the neighborhood consensus criterion. Note that the set of locations is not connected. Also, in the larger open space (left column: Market place), the place fields tend to be larger than in smaller places (right column: Street crossing "Krumme Brücke").

2.1 Feature detection

Micro-snapshots are defined as "upright speeded-up robust features" (U-SURF), as implemented in the OpenCV computer vision library [2, 3]. SURF finds interest points as intensity blobs by searching local maxima of the determinant of the image Hessian; color information is ignored. Scale invariance is achieved by considering each feature point at its optimal scale. In a second step, a 64-dimensional vector ("descriptor") is associated with each blob, containing information about image intensity gradients in a small patch around the interest point. The descriptor is used to compare and match features with each other. In U-SURF, it is not assigned a unique orientation and is therefore not rotation invariant. Rotation invariance is not required in our algorithms since the agent is confined to movements in the plane. The number of scale levels was limited to two octaves with two layers each since information about the viewing distance of a feature should not be completely ignored. SURF feature robustness was further increased by considering only features that appear in two successive frames. Whenever we refer to features as extracted from a given frame it means that those features also appeared in the preceding frame.

The features of a frame were ranked according to the value of the determinant of the local Hessian, i.e. their contrast. Then, the features were pruned so that only up to 30 features per frame were used for further analysis. In principle, the amount of features per frame could also be reduced globally by increasing the detection threshold of the SURF method. However, this could potentially lead to situations where no feature would be detected at locations where contrast is low.

We denote the features as f_i and their descriptors as $\boldsymbol{d}_i$;. $F = \{f_i \,|\, i = 1, \ldots, n\}$ is the set of all features stored in the system.

2.2 Feature matching

Whenever a feature is detected by the U-SURF procedure, it is checked for identity with all stored features in F using two criteria. First, the root mean squared difference between the descriptors of the compared features should be below a threshold ϑ_S. Second, to avoid aliasing in large sets of features, we require that the features share a context of at least ϑ_N other features (neighborhood consensus). To this end, we store for each feature f_i the set N_i of simultaneously visible other features. Two features f_i, f_j are thus identified with each other, if $\|\boldsymbol{d}_i - \boldsymbol{d}_j\|^2 < \vartheta_S$ and $|N_i \cap N_j| \geq \vartheta_N$. If an encountered feature is found to be novel, it is included into F.

The threshold for neighborhood consensus, ϑ_N, depends on the total number of features detected in each image. In our simulations, the value was set to $\vartheta_N = 4$ at up to 30 different features per frame. Note that aliasing still occurred occasionally even with expanded feature-neighbor matching (see section 2.3 and Fig. 4 below). In practice, the algorithm is robust against a small amount of outliers and can find and navigate routes even with faulty map data. See sections 2.5 and 3 for more details.

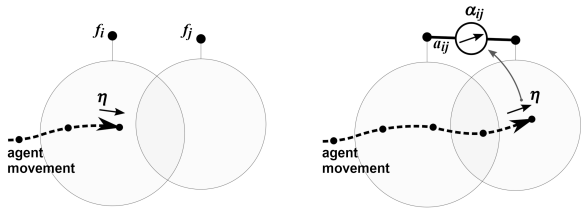

Fig. 3. Graph edge learning. Left: The dotted line shows the trajectory of agent with time steps (small dots) and learning steps (bold dots). Two features $\boldsymbol{f}_i$ and $\boldsymbol{f_2}$ have already been encountered and added to the feature set. The agent is currently moving with heading $\boldsymbol{\eta}$ in the place field of feature $\boldsymbol{f}_1$, but outside of the place field of feature $\boldsymbol{f}_2$. Right: The agent has now passed the overlap zone where both features are detected. At the next learning step, feature $\boldsymbol{f}_2$ is detected but feature $\boldsymbol{f}_1$ has moved out of sight. In this situation, a bidirectional pair of edges a_{ij}, a_{ji} is added to the graph and the edges are labeled with the current heading or its inverse, $\pm\boldsymbol{\eta}$.

Fig. 2 shows two features in the respective images from the Virtual Tübingen data set. In the second row, the position from which each feature was first defined and added to F is marked by a cross. For all positions in open space, color indicates the similarity of the most similar visible feature with the stored one; dark blue marks locations inside of houses that cannot be entered by the agent. The third row of Fig. 2 shows the area from which the feature is detected, using the two-step comparison procedure with similarity of descriptors and neighborhood consensus. It will be called the place field of the feature and roughly corresponds to the catchment areas in snapshot homing or the firing fields of a neuron tuned to the feature.

2.3 Graph edge formation

If a feature that was previously visible gets out of sight, the agent must have traveled a path out of the place field of this feature to some point inside the place fields of other features that remain or have become visible. This is the basic idea of learning graph edges in the algorithm. In order to avoid too high densities of graph links, new edges are not stored at every time step, but at a slower pace.

The basic time step of the algorithm is the frame, i.e. the recording of one image; we denote frames by the index t. The frame rate used in the graphics simulations below is 30 frames per second. Graph learning does not occur at every time step, but only once in a while, when the agent has moved sufficiently far away from the last learning event. Learning steps are counted by a second counter l. In the simulations below, the distance that the agent must have traveled before a new learning step occurs was set to two simulated meters, which corresponds to 30 frames. Note that we use the position ground truth of the VR simulation for stepping l. This can easily be relaxed by some simple path integration algorithm

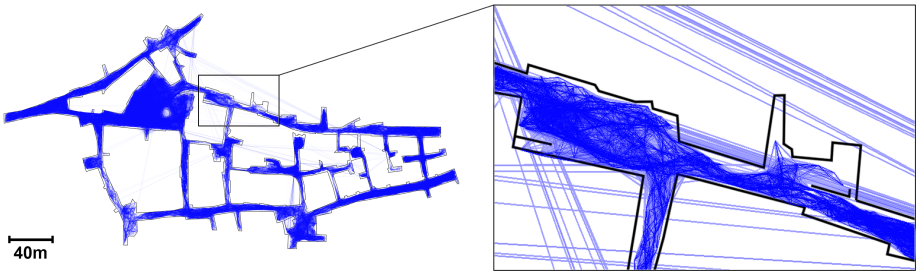

Fig. 4. Map of the testing environment "Virtual Tübingen" with view graph. The view graph can be embedded into a map by placing each feature at the agent's position from where it was first detected, and drawing the graph's edges between them (blue lines). The shown graph completely maps the virtual environment and consists of 222,433 nodes and 3,492,096 edges. Some of the edges connect very distant features (long blue lines crossing the empty white space). These are wrong connections resulting from aliasing.

which was, however, not implemented. The time steps at with learning counter l is stepped, are denoted by $t(l)$.

Let F_1 and F_2 be the sets of features visible at two subsequent learning steps l and $l+1$, respectively. Assume $f_j \in F_1$ and $f_j \notin F_2$. We then add a pair of directed edges a_{jk}, a_{kj} (forward and backward) between f_j and up to three randomly chosen features $f_k \in F_2$ to the graph. The edges are labeled with the current heading or its inverse, respectively (see Fig. 3 and next paragraph). The number three of edges created per vanishing feature is chosen to avoid exceedingly high computation costs in later graph search. For the same reason, the upper limit of a node's degree after repeated visits is set to 100.

An example of the graph after prolonged exploration appears in Fig. 4.

2.4 Edge labeling and reference direction

During the entire travel, the agent is estimating and maintaining an allocentric reference direction ν which is initialized to the value $\nu = 0$ at frame 1 (see Fig. 5). All other angles are expressed relative to this reference direction, i.e. in an allocentric scheme similar to the head direction in allocentric path integration [6, 21]. The dependent angles are (i) the current heading angle η_i, (ii) the feature bearings $\hat{\beta}_i$ stored with each feature f_i upon definition of the feature, and (iii) the directional labels of the edges a_{ij}, a_{ji} which are initialized with or against the current heading angle, i.e. $\alpha_{ij} = \eta_t$ or $\alpha_{ji} = \eta_t + \pi$, respectively.

The reference direction is constantly affected by a noise process $\boldsymbol{n}$ and updated according to the available landmark cues, i.e. the bearings of known features. Let F_t denote the set of known features visible at frame t and $\hat{\beta}_i$ be their stored bearings. The agent compares the current feature bearings with the stored ones and computes the average deviation as a circular mean. Then, the reference

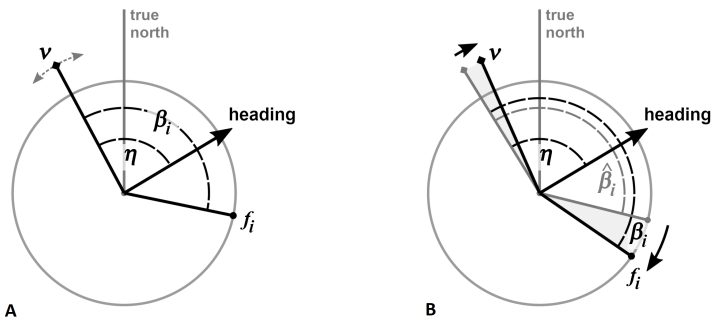

Fig. 5. The head-direction system. A: During the entire simulation, the system is maintaining a reference direction ν which is initialized to the movement direction in the first frame.Heading angle η and feature bearings β_i are always expressed relative to ν. The "true north" direction is known to the virtual reality simulation, but not to the agent. B: If a feature is detected, its stored bearing label $\hat{\beta}_i$ is compared to the actual bearing in the current image, β_i and the reference direction is updated so as to reduce the difference between $\hat{\beta}_i$ and β_i. Of course, this is done for many features simultaneously, as described in Eq. 1.

direction is updated as

$$\nu_{t+1} = \nu_t + \frac{\lambda}{|F_t|} \underset{\{i|f_i \in F_t\}}{\text{cmean}} \left(\hat{\beta}_{t,i} - \beta_{t,i}\right) + \boldsymbol{n}, \tag{1}$$

where λ is set to 0.05 and the standard deviation of $\boldsymbol{n}$ is set to $\sigma = 0.025$ rad. The circular mean of a set of angles $\{\gamma_i\}$ is defined as

$$\underset{A}{\text{cmean}}(\gamma_i) := \text{atan2}\left(\sum_{i\in A} \cos\gamma_i, \sum_{i\in A} \sin\gamma_i\right). \tag{2}$$

This updating rule attributes the average bearing error to the reference direction. It can compensate for the noise, but introduces a new type of error if the features are unequally distributed in the image. Assume, for example, that the agent relies only on features on its left. If it moves forward, these features will move further to the left, leading to positive deviations $\hat{\beta}_i - \beta_i$. The algorithm will then assume that the reference direction has turned to the left. As a result, the reference direction in a large environments drifts with the agent's position, as is illustrated in Fig. 6. However, in prolonged exploration, the assumed reference directions convergence to a stable, locally consistent distribution over explored space.

In addition, the stored bearings for each feature, $\hat{\beta}_i$ are updated at each learning step at which the feature i is re-detected by the iterative mean:

$$\hat{\beta}_{t(l+1),i} = \frac{c_i}{c_i+1}\hat{\beta}_{t(l),i} + \frac{1}{c_i+1}\beta_{t(l+1),i}, \tag{3}$$

where c_i is a counter stepped at each update and $\beta_{t(l),i}$ is measured relative to the current compass direction ν_t.

Finally, a link a_{ij} may be rediscovered upon a later encounter of the same location. In this case the associated direction label α_{ij} is updated as

$$\alpha_{ij}^{\text{new}} = \frac{c_{ij}}{c_{ij}+1}\alpha_{ij}^{\text{old}} + \frac{1}{c_{ij}+1}\eta_t. \tag{4}$$

Again, this can happen only when the slow learning counter l is stepped. As for the bearings, c_{ij} is a counter stepped at every update and the η_t is measured from the current compass direction. Note that the counters c_i and c_{ij} in Eqs. 3 and 4 can be avoided by replacing the bearing and heading angles by unit vectors and storing sums of these unit vectors as labels. From the accumulated vectors, the angles can then simply be obtained by the atan2 function.

Fig. 6. Compass direction drift over a large explored area. The ν estimate may deviate substantially (over 90 degree) from its starting value, but remains locally consistent.

After training, the agent will have built a data structure $\{F, E\}$ where F is a set of features f_i with descriptors $\boldsymbol{d}_i$, expected bearings $\hat{\beta}_i$, and feature contexts N_i; the subset of currently detected features F_i characterizes the position of the agent. E is a set of directed edges a_{ij} with direction labels α_{ij} indicating the direction of movement required to get from the place field of feature i into the place field of feature j. In addition to this stored data, a reference direction ν is maintained as a working memory and updated from the comparison of known and perceived feature bearings. This latter system models the head-direction systems of rodents and flies [21, 19]. Place recognition is based entirely on the recognition of features and the associated place fields.

2.5 Pathfinding and voting

The algorithm presented here is able to navigate the mapped environment by using graph search methods on the view graph. It is able to guide an agent to any user-selected goal location, as long as the environment has been explored sufficiently. The goal location is defined by a set of known features, and can for example be provided by an image depicting the goal. The algorithm then calculates multiple non-overlapping paths from features at the agent's current position to the set of goal features, and uses a voting scheme to obtain navigable trajectories the agent can follow towards the goal location (Fig. 7).

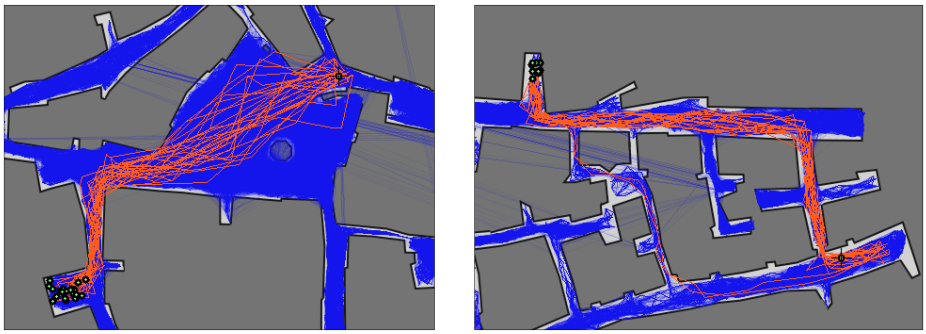

Fig. 7. Route examples. Two different examples of path bundles (orange) from the agent's current position (black cross) to a goal locations (set of green dots). The blue background shows the edges of the view graph, as described in Fig. 4.

In each pathfinding event, for one currently visible feature, the shortest path is found to one of the features in the goal set with Dijkstra's algorithm [8]. Then, the nodes of that path are temporarily removed from the graph, except for the first and last nodes, and the search is repeated for another randomly selected pair of nodes. Due to the node removal, each path will have zero overlap with all previous paths. Still, when represented in a metric map, path trajectories will be similar due to overlapping place fields.

The search terminates when the pair of randomly selected start and goal nodes is unconnected in the graph lacking the temporarily removed nodes or when an upper limit has been reached. For example, in the tests detailed in the "Evaluation" section below, we used 30 successive Dijkstra searches, but the algorithm regularly found only a lower number of routes (~ 27), depending on the amount of exploration. Note that all edges are considered to have the same length, i.e., Dijkstra paths only differ in the number of edges they traverse.

Once a bundle of paths has been obtained, we could use the initial edge labels α_{ij} to determine the movement direction from the start location. Later however, it is not clear which step of each path applies at each position along the overall travel. Therefore, at each position, we determine the set of currently

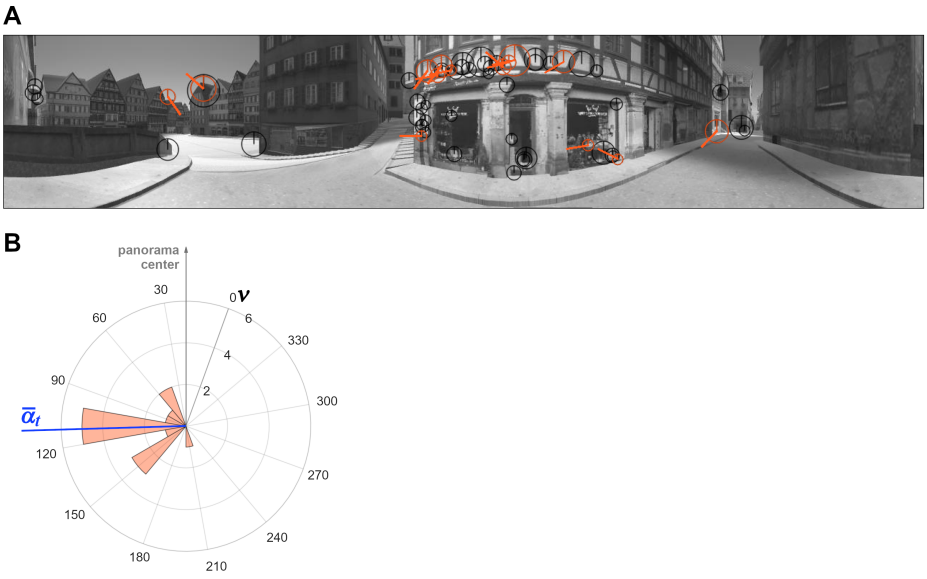

Fig. 8. Direction voting. 360 degree panorama frame with detected SURF features (black and orange circles with vertical bar). During pathfinding, movement is derived from features that are also part of the path bundle (orange circles): The thick lines originating from the orange features show their respective movement direction vote relative to 0 degree (thin vertical stripe). The histogram shows the votes sorted into 10 degree bins and the mean direction $\overline{\alpha}_t$.

visible features also included in the present bundle. From each such feature we take the next edge along the respective path and thus obtain a set of movement votes (Fig. 8).

Each Dijkstra path p in the bundle P has an ordered set of edges $E_p = \{a_{ij}, a_{jk}, a_{kl,\ldots}\}$ and set of nodes $F_p = \{f_i, f_j, f_k, f_l, \ldots\}$. At navigation time, the set of visible features is F_t. We now consider the indices of all outgoing edges of currently visible features contained in a path of the bundle, $J_t = \{(i,j) \mid f_i \in F_t \wedge a_{ij} \in \bigcup_{p \in P} E_p\}$. The set of locally applicable motion directions is then given by $\{\alpha_{ij} \mid (i,j) \in J_t\}$. From these we obtain the movement consensus as

$$\overline{\alpha}_t = \operatorname*{cmean}_{J_t} \alpha_{ij}, \tag{5}$$

where cmean is the circular mean as defined in Eq. 2.

The final heading vector $\boldsymbol{\eta}_{t+1}$ is calculated with stiffness $\kappa \in [0,1]$ as

$$\boldsymbol{\eta}_{t+1} = \kappa \boldsymbol{\eta}_t + (1-\kappa)(\cos\overline{\alpha}_t, \sin\overline{\alpha}_t)^\top. \tag{6}$$

This results in a smoothing of the trajectory to reduce sway and reduce corner-cutting behavior; κ was set to 0.7.

Moving into the direction of $\boldsymbol{\eta}_{t+1}$ ideally leads the agent along a route specified by the bundle of paths, where it will continue to encounter labeled nodes.

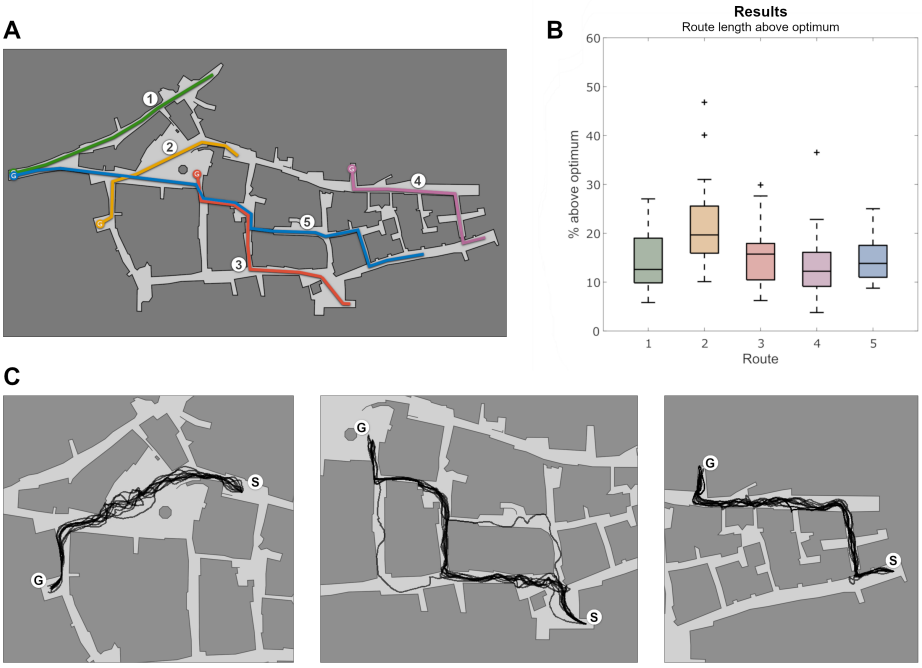

Fig. 9. Performance in "Virtual Tübingen". A: Map depicting the evaluation routes. B: Results of the evaluation. The box plot shows route length above an optimal trajectory, as depicted in A. Performance is somewhat worse for route 2, because it traverses a wide open area containing lots of distant landmarks, which are worse for exact navigation. C - E: Ten repetitions of each of the routes 2, 3 and 4. C: repetitions of route 2 show larger variations in open spaces. D: repetitions of route 3 show occasional choice of route alternatives as well as directional sway within single repetitions. E: repetitions of route 4 show the lowest variability among the evaluation routes

To facilitate this process, during path following, the number of features detected in each frame is doubled, which improves the odds of detecting labeled nodes. If the number of usable nodes, $|J_t|$, drops below a threshold of two, we assume that the agent has diverged from the path. In this case, a new bundle of Dijkstra paths is calculated with the current feature set F_t as a starting point.

3 Evaluation

Our procedure differs in three important respects from standard approaches to graph-based navigation such as [10, 18, 9, 11]. First, the state of the agent is not characterized by a single node of the graph, but by a set of nodes which is unique for each confusion area. Second, path planning, i.e., the process of generating a path from the map, is not a single graph search but a two-step process involving

a bundle of graph paths and a local voting scheme for direction. Third, we do not employ an explicit control law or homing scheme for approaching intermediate goals. Indeed, such intermediate goals are not explicitly used. Rather, the agent proceeds in small, "ballistic" steps.

The algorithm determines if the goal is reached by comparing the set of currently visible features, F_t, to the set of goal features, F_g, and considers the goal to be reached if $|F_t \cap F_g| \ / \ |F_g| \geq 0.35$. Note that there may be some offset between the agent's final position and exact goal location since there is no optimization step in our ballistic procedure.

Relying on the average of a set of movement instructions solves the problem of wrong connections introduced into the graph due to aliasing, if enough alias-free paths are present. A crucial problem of mapping without global metric embedding and only relying on visual similarity is aliasing, the possibility that two features at distinct locations are confused. This can lead to the formation of wrong or impossible connections in the graph, which tend to be shorter than navigable connections (see Fig. 4). However, as long as a sufficient number of correct edges corresponding to navigable trajectories exist, the votes of the erroneous connections will not cause navigation to fail.

Finally, if the agent is unable to move during navigation, for example due to obstacles or being stuck in a corner, or if no consensus can be found in the set of movement instructions, the bundle of paths is recalculated, which has always solved the problem in our simulation. If the agent ever gets lost, for example because no known features are recognized, it may return to exploration behavior for a short while (e.g., random walk)

The algorithm was tested and evaluated in a virtual environment of the downtown area of Tübingen, Germany (3D model based on [24]), rendered in the Unity engine[1]. The agent in the virtual environment was equipped with a 360 degree horizontal FoV and 60 degree vertical FoV camera projecting to a 1280×240 pixel image. Depending on location, the SURF feature detector would detect some 20 to 350 features per image, which were pruned to a maximum of 30 during exploration and 60 during path following.

When the agent had explored every street of the model at least once, exploration was terminated (compare Fig. 4). In the subsequent test phase, five pathfinding tasks were defined as start and goal views. Each task was repeated 20 times, and the traveled distance was measured and compared to that of the shortest possible route (Fig. 9A). The algorithm solved the tasks by first selecting features from the start view and estimating the reference direction ν from the features' bearings. Next, 30 Dijkstra paths were calculated, and from these, the overall trajectory was generated as described above.

The trajectories found for a single task may greatly differ between repetitions due to many stochastic influences, such as node selection for start and goal nodes and the noise added to the reference direction update. Further variation is introduced by numerical effects in collision detection and the latency between concurrent components of the programs running the algorithm and simulation.

[1] Unity Technologies, Unity 2018.1.5f1, https://unity3d.com/, 2018

The algorithm may even guide the agent along different roads over multiple trials if they are close in length to the optimal route (see Fig. 9D).

The algorithm managed to successfully and efficiently guide the agent from start to goal in all trials with steady directional movement. On average, the agent's routes were only 10% to 20% longer than the optimal routes (Fig. 9B). The agent performed better, i.e., the routes were shorter, when they were leading mostly through roads and alleys rather than traversing large open spaces such as the market square. The algorithm was able to guide the agent even with large drift in the reference direction ν of over 90 degree offset from the starting ν (see Fig. 6).

In conclusion, double population coding successfully combines the idea of spatial population coding with topological navigation by stimulus-response associations. It does not require highly processed input information but works with sequences of raw panoramic snapshots and basic feature extraction. Navigational performance is overall good. With respect to the question of parsimony, systematic work on the number of required features and graph links is required. In future work, the algorithm will be used to model human navigational performances such as the perception of local reference directions [15], view voting [14], or local metric in spatial long-term memory [25].

References

1. Baddeley, B., Graham, P., Husbands, P., Philippides, A.: A model of ant route navigation driven by scene familiarity. PLoS Computational Biology **8**(1) (2012). https://doi.org/10.1371/journal.pcbi.1002336
2. Bay, H., Ess, A., Tuytelaars, T., Van Gool, L.: Speeded-up robot features (SURF). Computer Vision and Image Understanding **110**, 346–359 (2008)
3. Bradski, G.: The opencv library. Dr. Dobb's Journal of Software Tools (2000)
4. Cartwright, B.A., Collett, T.S.: How honey bees use landmarks to guide their return to a food source. Nature **295**, 560 – 564 (1982)
5. Cartwright, B.A., Collett, T.S.: Landmark maps for honeybees. Biological Cybernetics **57**, 85 – 93 (1987)
6. Cheung, A., Vickerstaff, R.: Finding the way with a noisy brain. PLoS Computational Biology **9**(11) (2010), e112544
7. Differt, D., Stürzl, W.: A generalized multi-snapshot model for 3D homing and route following. Adaptive Behavior (2020). https://doi.org/10.1177/1059712320911217
8. Dijkstra, E.W.: A note on two problems in connexion with graphs. Numerische Mathematik **1**, 269 – 271 (1959)
9. Franz, M.O., Schölkopf, B., Mallot, H.A., Bülthoff, H.H.: Learning view graphs for robot navigation. Autonomous Robots **5**, 111 – 125 (1998)
10. Kuipers, B.: Modeling spatial knowledge. Cognitive Science **2**, 129 – 153 (1978)
11. Kuipers, B.: The spatial semantic hierarchy. Artificial Intelligence **119**, 191 – 233 (2000)
12. Madl, T., Chen, K., Montaldi, D., Trappl, R.: Computational cognitive models of spatial memory in navigation space: a review. Neural Networks **65**, 18 – 43 (2015)

13. Mallot, H.A., Basten, K.: Embodied spatial cognition: biological and artificial systems. Image and Vision Computing **27**, 1658 – 1670 (2009)
14. Mallot, H.A., Gillner, S.: Route navigation without place recognition: what is recognized in recognition–triggered responses? Perception **29**, 43 – 55 (2000)
15. Mou, W.M., McNamara, T.P.: Intrinsic frames of reference in spatial memory. Journal of Experimental Psychology: Learning, Memory and Cognition **28**, 162 – 170 (2002). https://doi.org/10.1037//0278-517 7393.28.1.162
16. Muller, R.U., Stead, M., Pach, J.: The hippocampus as a cognitive graph. Journal of General Physiology **107**, 663 – 694 (1996)
17. O'Keefe, J., Nadel, L.: The hippocampus as a cognitive map. Clarendon, Oxford, England (1978)
18. Schölkopf, B., Mallot, H.A.: View–based cognitive mapping and path planning. Adaptive Behavior **3**, 311 – 348 (1995)
19. Seelig, J., Jayaraman, V.: Neural dynamics for landmark orientation and angular path integration. Nature **521**, 186 (2015)
20. Smith, L., Philippides, A., Graham, P., Baddeley, B., Husbands, P.: Linked local navigation for visual route guidance. Adaptive Behavior **15**(3), 257–271 (2007)
21. Taube, J.: The head direction signal: Origins and sensory-motorintegration. Annual Reviews in Neuroscience **30**, 181 — 207 (2007)
22. Tolman, E.C.: Purposive behavior in animals and men. The Century Co., New York (1932)
23. Trullier, O., Wiener, S.I., Berthoz, A., Meyer, J.A.: Biologically based artificial navigation systems: Review and prospects. Progress in Neurobiology **51**, 483 – 544 (1997)
24. Veen, H.A.H.C.v., Distler, H.K., Braun, S.J., Bülthoff, H.H.: Navigating through a virtual city: Using virtual reality technology to study human action and perception. Future Generation Computer Systems **14**, 231 – 242 (1998)
25. Warren, W.H.: Non-Euclidean navigation. Journal of Experimental Biology **222** (2019). https://doi.org/10.1242/jeb.187971
26. Wiener, J.M., Shettleworth, S., Bingman, V.P., Cheng, K., Healy, S., Jacobs, L.F., Jeffrey, K.J., Mallot, H.A., Menzel, R., Newcombe, N.S.: Animal Navigation. A Synthesis, pp. 51 – 76. The MIT Press (2011)
27. Wilson, M.A., McNaughton, B.L.: Dynamics of the hippocampal ensemble code for space. Science **261**, 1055 – 1058 (1993)