IET Computer Vision

Special Issue: Computer Vision for Smart Cameras and Camera Networks

IET Journals
The Institution of Engineering and Technology

Decentralised indoor smart camera mapping and hierarchical navigation for autonomous ground vehicles

ISSN 1751-9632
Received on 1st December 2019
Revised 19th July 2020
Accepted on 17th September 2020
E-First on 21st October 2020
doi: 10.1049/iet-cvi.2019.0949
www.ietdl.org

Taylor J.L. Whitaker[1] ✉, Samantha-Jo Cunningham[1], Christophe Bobda[1]
[1]*Electrical and Computer Engineering, University of Florida, Gainesville, FL, USA*
✉ *E-mail: t.whitaker@ufl.edu*

Abstract: In this work, the authors propose a novel decentralised coordination scheme for autonomous ground vehicles to enable map building and path planning with a network of smart overhead cameras. Decentralised indoor smart camera mapping and hierarchical navigation supports the automatic generation of waypoint graphs for each camera in an environment and allows path planning through the environment across multiple camera fields of view, or subviews. The proposed solution utilises the growing neural gas algorithm to learn the topology of unoccupied working space in each subview for maintaining a dynamic waypoint graph on each camera. The authors' pathing solution leverages a modified version of the A* algorithm to compute paths in a decentralised and hierarchical fashion. Waypoint generation was simulated and analysed on a generated environment to ensure it is both effective and efficient, while path planning was simulated on various randomised hierarchical graphs to effectively compare the proposed Decentralised-A* (D-A*) algorithm against standard greedy search. The proposed method efficiently handles the cases where other robot navigation methods are otherwise weak and ineffective, while still providing avenues for further optimisation of resource overhead for both the smart camera network as well as the robots themselves.

1 Introduction

Indoor robot navigation has become a frequently researched area due to the increasing benefit of automation to augment jobs within warehouses, factories, and retail facilities, in addition to many other areas. Autonomous navigation is key to successful deployment in these environments as the benefits increase with minimising manual human interaction with the robots and corresponding coordination system.

Parker [1] categorises multi-robot navigation schemes in literature into two paradigms, coupled and decoupled approaches. As navigation is a composite problem including localisation and path planning, both of which can be further broken down, coupled approaches centralise the management of the individual components while decoupled approaches separate components to independent elements. Coupled approaches have the benefit of operating on the complete set of information available of an environment, while decoupled approaches distribute the problem to units that may not have a complete observation of the environment. An example of a coupled approach would be performing all map building and path planning on a central server which collects observational data from environmental and robot sensors and distributes instructions to each robot. This centralised server could allow for optimal pathing for every robot with the complete set of observational data but suffers from the extreme overhead growth associated with scaling the number of robots and size of the environment. A central server introduces an additional consequence as any downtime suffered by the server limits or entirely halts all path planning and thus the robots themselves. A decoupled approach is exemplified by any such system that distributes the localisation/mapping and path planning to separate units, such as the robots themselves, each of which only being aware of only locally collected observations of the environment. This approach puts a heavy load on the individual units and requires that unnecessary resources be dispatched to each, increasing the overhead of size, weight, and power constraints of each as well. Downtime is not as much of a concern for decoupled approaches though cannot be considered complete as operations are only performed on subsets of observation data.

This work aims to contribute a viable solution for coordinating an arbitrary number of autonomous ground vehicles (AGVs) in an arbitrary and dynamic indoor working environment, one which taps into advantages of both coupled and decoupled approaches. We present a system in which a hierarchical graph is continually updated and used within a decentralised system of smart overhead cameras to allow path planning in an environment. Our system enables the sharing of only relevant data between smart cameras to increase global awareness at each node while still minimising overhead growth as cameras or robots are introduced.

The decentralised indoor smart camera mapping and hierarchical navigation (DISCMAHN) system embodiment can be illustrated above with the overhead smart cameras establishing subviews of an environment (Fig. 1). Individual fields of view can be seen with the dotted boundaries. The collection of subviews provides a global vision system and each camera is capable of routing AGVs through the environment around obstacles.

Our work on DISCMAHN is organised as follows. In Section 2, we cover works with contributions relevant to robot coordination. We detail our proposed solution and its components in Section 3. An evaluation is performed for our waypoint generation and path planning strategies in Section 4. Section 5 concludes our work.

2 Related works

As stated in the previous section, coordinating one or more robots is a composite solution that includes the localisation of robots or agents in an environment in addition to planning individual paths. Localisation typically implies there is some representation of the environment or map in which the agents must be assigned a position and orientation. The map can be in many forms from a simple static image of the environment, a graph of distinct points with connecting edges, or some alternative data structure suitable for maintaining a static or dynamic map. Maps can be one of many cardinalities though we only consider those of two dimensions as they are most relevant to our proposal.

Dynamic maps can be produced with a variety of techniques and with the aid of one or many sources. Simultaneous localisation and mapping (SLAM) [2] is the standard solution to changing environments. It requires agents to hold some ability to perceive

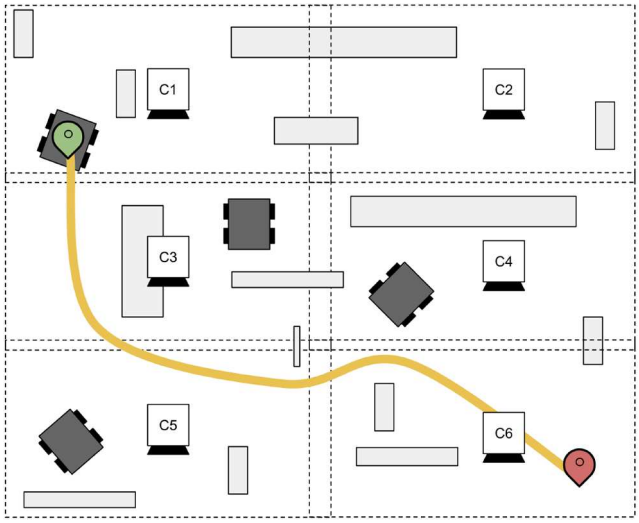

Fig. 1 *DISCMAHN illustration*

their surroundings for estimating state. As agent perception does not span the entire environment, performing SLAM locally leaves agents unaware of the changes in the environment beyond its relative observational regions. This can be combated with a global SLAM instance running on a central server with access to all agent's observations, and applied to indoor environments, can be quite successful. Though, SLAM falls short of being unable to provide constant observation of the entire working area without a particular saturation of agents.

A camera is an alternative solution that can be suitable for localising objects within its field of view and, if placed correctly, could provide a method of generating a map. The work of [3, 4] both utilise a robot tagging method for determining robot localisation and identification within a captured image. This is done with a single camera placed in the environment. These works can be helpful for coordination, though do not support the generation of a map representation and no transform is available from the image coordinate frame to the environment's. Utilising multiple cameras in an overhead configuration could allow for observing an entire environment, suitable for both localisation and mapping. Real-time localisation is performed with a global vision system of overhead cameras in [5, 6] where soccer-playing robots are tracked with digital cameras and their poses are determined. The cameras provide coverage of the entire playing field and thus can utilise the camera calibration metrics and positioning to determine poses in the coordinate frame of the field. Both works centralise the image processing of the cameras to an attached server that would not scale well with the addition of more cameras and would require increased processing power for tracking every introduced robot. Moreover, in this application of robot soccer, tracking is performed almost exclusively on an image of the field that does not contain many obstructions, permitting the lack of a true map representation for such a system. Introducing a global vision system to larger environments requires additional consideration for resource distribution, as well as scalable path planning, factors the above solutions fail to address.

Separate from localisation is path planning of agents, which is often constrained by the map representation of the environment they operate in. A* being a largely adopted solution capable of finding optimal paths in a graph rather quickly is one such algorithm that can either flourish or suffer depending on how the environment is represented as a graph. The A* algorithm runs in $O(|E|)$ time and requires $O(|V|)$ space in the worst-case scenario, where V is the set of vertices and E is the set of edges in the graph [7]. With this, it is evident that increasing the graph size and connectivity will have an impact on its performance. As an example, a global view of stitched camera images of an environment could easily be represented as a graph with pixels as vertices and neighbouring pixels identified with undirected edges. Increasing the size of the environment would lead to a need to increase the coverage of cameras to accurately represent an accurate environment and thus increase the number of pixels and the resulting graphs. Scaling in this fashion is not feasible for a standard A* implementation. Overcoming this downfall with A* can be combatted in several ways with many simply focusing on reducing the size of the graph on which it searches for paths. Zhong *et al.* [8] used a modified A* path planning algorithm that segments the working region into a grid to path from the starting local region (fine) to the upper nodes (coarse) to reduce the time to calculate a path. The inverse approach was proposed in [9]. Hierarchical graphs for A* can facilitate a speedup though suffer a loss of optimality. This is permissible if the loss of optimal pathing can be outweighed by the benefits of speedup of path calculation. These approaches enable a regionalised map but do not consider the process by which their map representation can be automatically generated.

To facilitate fast path planning in arbitrary environmental topologies, we sought a method that might learn a graph approximation of navigable space. The work of Felder *et al.* [10] attempts to do this with waypoint graphs, though falls short with achieving a feasible approximation of free working space. One such algorithm for approximating topology is the growing neural gas (GNG) network, presented in the work of Fritzke [11]. Given a set of training data, this algorithm will grow and prune a graph to fit the data set. With the ability to tune the parameters of growth, pruning, and maximum size, this method can accurately learn the data or environment while still enabling a reduction of dimensionality from the original data set with its approximation. Take for instance a two-dimensional map or grid of points representing free and occupied space in an environment. The subset of points in unoccupied space could be provided as a training data set and thus the topology of free space could be learned and the graph representation's vertices used as navigational waypoints. Tuning the number of waypoints would allow the tuning of path-planning performance on the graph and speedup compared to a path planning on the original data set. The work of Tence *et al.* [12] recognises the importance of having no preconceptions of an environment that is dynamic, especially in their case of video games. Their work uses the GNG algorithm for teaching agents how to navigate the game environment and uses the help of a human player for building a graph of navigable space. This supports the ability for flexible coordination of agents, though suffers from the need for human interaction to learn the environment first. Automatically generating navigation waypoints with the GNG algorithm and without human interaction is proposed in [13]. This work shows much success with learning the topology of navigable space in generated mazes and SLAM-generated maps and even makes several improvements to the original algorithm for better supporting graphs serving as waypoint graphs. Such improvements include adding a stopping criterion to the GNG algorithm and scaling the graph node generation rate for more sparse graphs with still competitive approximations of the data set. A graph that represents an environment with any accurate approximation is an avenue towards efficient path planning. Unfortunately, Dellinger *et al.* [13] does not employ any method for dynamic situations.

All of the above methods have taken positive steps towards viable navigation schemes for agents in a dynamic environment. However, none of these methods provide details about the methods of coupling these features together for a scalable coordination scheme. We aim to extend on the works above with a pipeline that can incorporate many of their advances. To the author's best knowledge, no published works have presented the use of smart overhead cameras to generate dynamic waypoint graphs that facilitate decentralised and hierarchical path planning across the camera network. The details of our coordination scheme are outlined in the proceeding sections.

3 Decentralised indoor smart camera mapping and hierarchical navigation

This work aims to contribute towards coordinating an arbitrary number of AGVs in an arbitrary and dynamic indoor working environment. We propose DISCMAHN as a solution. We leverage

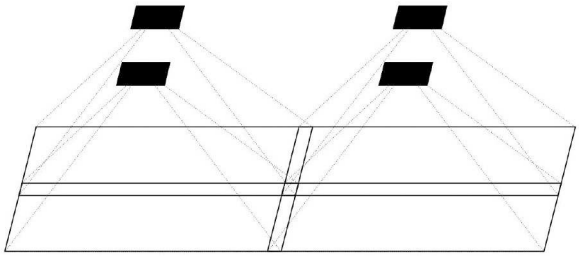

Fig. 2 *Grid arrangement of smart overhead cameras*

an array of fully decentralised overhead-mounted smart cameras deployed in an environment that together provide a global vision system. With the constant observation of the working area, DISCMAHN can facilitate resilience to any changes in the layout and produce a dynamic map approximation on which path planning can be performed. Coordinating multiple robots requires multiple instances of path planning to be carried out, simultaneously if efficiency is a target as in our system. With individual cameras managing the monitoring of separate regions, we propose assigning individual path-planning tasks to the cameras themselves as the need arises. The selected camera chosen for the task for any given robot that may be requesting a path is determined by the robot's starting location. Pathing requests are issued to the camera network with a navigable path being returned from the camera with visibility of the requesting robot in its field of view. This navigable path, issued by a single camera, spans the environment and multiple camera fields of view as determined by the destination. This is done by allowing communication between all nodes of our system; each camera and robot. Distributing map building and path planning in a decentralised camera network allows resources to be optimised across all smart cameras and can reduce the overhead imposed on individual robots.

Summarising our contributions with this work, we:

- Formally describe a novel smart camera network that is suitable for deployment in an environment requiring robot coordination throughout.
- Establish a process by which waypoint graphs can be automatically generated for each camera.
- Formalise the organisation scheme of cameras and subsequent individual waypoint graphs.
- Propose decentralised path planning on a distributed hierarchical graph of waypoints.
- Provide an analysis of the performance of waypoint generation and path planning.

3.1 Assumptions

The proposed coordination system requires that all components can establish direct communication with any other. Thus, we assume there is some middleware in place to facilitate this. Most important is the requirement of communication between any two cameras in the network. As will be discussed further, cameras must be able to exchange information requests during path planning. Middleware for robotic applications communication is common, with a largely adopted solution being the robotic operating system (ROS) [14]. The application of ROS to our work is of interest to the authors, though steps beyond the focus of this work as it does not directly offer a decentralised communication scheme out of the box. As we target AGVs, we must additionally define the minimum capabilities imposed on each robot by our coordination system. Other than a communication middleware, the primary requirement is that AGVs are able to operate on velocity commands that are issued to them and subsequently move through the environment. It is also suggested that robots have the appropriate resources for basic local collision avoidance as a failsafe. These are minimal functionalities typically including under the umbrella term AGVs.

We do not cover the process by which a path is converted to a set of velocity instructions for the robots, as this requires localisation of robots within a captured image and a known transformation of the camera's coordinate frame to the environment's. There has been extensive work on robot localisation within a map and even an image [3, 4]. With the work discussed here, it is assumed that a transformation handler is in place. The tool-based ROS middleware [14] includes standard packages that also facilitate transformations between various frames of reference.

3.2 Global vision with smart cameras

We define an environment for our system which is dynamic and arbitrary. It may be an industrial storage facility or factory, a retail venue, or even an indoor public environment (Fig. 2). Our system deployment consists of a set of overhead smart cameras and a set of robots to be coordinated. The collection of smart cameras views the entire working region of the environment, where an individual camera can see a small subset of the global working region. We refer to a camera's field of view as its subview. Each camera has a subview that has a slight overlap with all of its neighbouring cameras. The overlap is vital as we can automatically establish valid transition points between any neighbouring camera's fields of view. Cameras are currently arranged in a grid structure to simplify this neighbouring region overlap determination and each camera is mounted at the same height for identical subview resolutions and coverage area. Realistically, the layout of cameras should allow for varying heights and coverage areas as different indoor environments will impose their own physical camera placement constraints. The provided examples of camera placement should not be perceived as limitations to the system, as the layout of cameras does not need to be a grid at all. Image stitching could be used to determine the topology of the camera array, while the waypoint generation and path-planning techniques need only to be aware of the neighbouring nodes' positions in the array to accurately establish appropriate transition regions and transition points. This is discussed further in Sections 3.3–3.5.

We define our set of smart cameras in the decentralised grid as $C = \{C_1, \ldots, C_i\}$, with i equal to the number of cameras required to provide full coverage of an environment. At any given time, each camera can produce a local subview image, S, such that

$$S = \left\{(x_1, y_1, \text{pixel}_1), \ldots, \left(x_j, y_j, \text{pixel}_j\right)\right\},$$

with j equal to the product of the raw image's resolution and pixel is a tuple representing value in the RGB colour space. In our proposed deployment, cameras will be arranged to force subviews to overlap, dictating

$$\forall\, S_a \in S, \exists\, S_b \in S \mid S_a \cap S_b \,!= \varnothing\,.$$

We propose a camera network such that each is aware of some global constants and can maintain a set of shareable local information. Every camera will have a known calibration and retain configuration information such as transition suitable borders. This enables each camera to determine its own set of appropriate transition regions, or regions where navigating to another subview is possible. Specifically for global constants, each camera should be aware of, or at least have some way of requesting, the grid layout to facilitate middleware communication between them. This layout must include relevant connection information for communication and is necessary for allowing each camera to synchronise with its immediate neighbours to determine grid features such as overlap and subsequently shared transition regions. This is further discussed in Section 3.5.

Elaborating on the smart cameras, we propose a FPGA SoC centric setup. That is, we propose a smart camera with the basic requirements as set by the assumptions in Section 3.1, a TCP/IP networking stack that is preferably Ethernet-based to accommodate high bandwidth and a suitable processing core to enable the ROS middleware. With regards to ROS, a Linux system is typical and thus requires a CPU on which the Linux kernel can be booted and an appropriate operating system can be run. We further propose an FPGA-based system as the authors foresee an opportunity to utilise programmable logic to accelerate some of the techniques discussed in the following sections in future work.

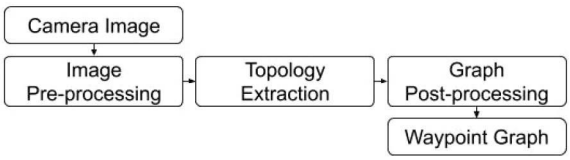

Fig. 3 *Generalised waypoint graph generation process*

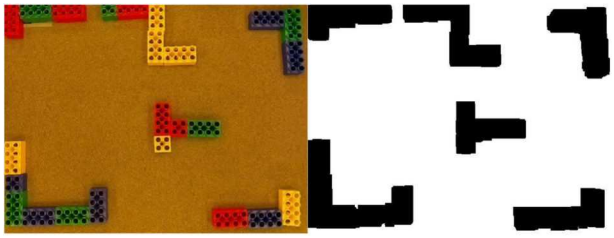

Fig. 4 *Lab-captured image (left) and resulting binary image of layout extraction (right)*

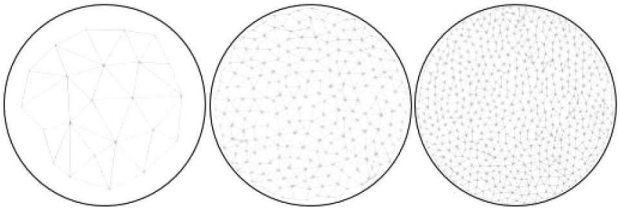

Fig. 5 *Example of GNG topology approximation of a circle*

In DISCMAHN, cameras maintain constant observation of their local working region. Each camera provides a raw image output of the environment within its subview. Real-time observation requires the cameras to provide either an image stream or a video stream, though we focus on a stream of individual images, here. The designation as a smart camera originates from the added local processing that occurs proceeding a capture. From a simple image, we propose a pipeline that can determine occupied and unoccupied space in a subview and automatically generate a graph approximation of unoccupied space to serve as a waypoint graph for local path planning. Additionally, Sections 3.4 and 3.5 discuss how each camera is capable of global path planning over numerous subviews of network cameras.

3.3 Automatic waypoint graph generation

With consistent surveillance of individual subviews, each camera is fit with the most useful observations of their region of an environment. Thus, we propose to compute a waypoint graph from a camera captured image for efficient subview-local navigation. We discuss a pipeline for a single camera image purposed for generating a graph approximation of the environment local to a subview. Modifications to the generalised process, in Fig. 3, to facilitate real-time updates to this graph approximation are discussed in Section 4.1.

3.3.1 Image pre-processing: We utilise mean shift, a technique for the reduction of multimodal feature space into segmented arbitrary clusters, described in [15], to get operable data of camera subviews. We use a histogram to reduce the results of the mean shift algorithm into a binary image, as shown in [16]. A subview S is used to produce an occupancy designation for all points, $S_{\text{occupancy}}$ (our binary image), where

$$S_{\text{occupancy}} = \{(x_1, y_1, \text{state}_1), \ldots, (x_j, y_j, \text{state}_j)\}$$

and *state* is equal to one of two values, zero or one, which can be interpreted as occupied or unoccupied, respectively. This mean shift method is robust in its ability to decipher ground from any obstacle. However, it requires the assumption that safe ground is the relative majority of image pixels. A truly resilient method of generating a binary image is a target of future endeavours. Fig. 4 demonstrates this method employed on a test image capturing a small region of blocks representing obstacles in our lab and shows adequate success at visualising the environment layout.

The goal of preprocessing images is to establish navigable space in the environment. With the resulting $S_{\text{occupancy}}$, the subview can be partitioned into data sets of points of free and occupied space. This most importantly produces a set of points S_{free}, where

$$S_{\text{free}} = \{(x_1, y_1), \ldots, (x_j, y_j)\}.$$

To work towards the goal of efficiently distributing computation throughout our camera grid, with a set of configuration constants, each camera is able to further partition its free space regions into disjoint regions suitable for transitions out of the current subview and into a neighboring camera's subview. In particular, given a border width, each camera can search for any free space on the outer edges of the subview that intersect with the candidate transition border region. With each camera being aware of its own possible transition regions, this enables any two cameras to coordinate and establish valid transition regions between their respective subviews. We define a candidate transition region as a gateway G, such that

$$G = \{(x_1, y_1), \ldots, (x_k, y_k)\}$$

with k as the number of points of contiguous points of S_{free} in the specified border region. Determining contiguity can be performed in numerous ways, though we simply look for the largest rectangle that may fit in this intersection. The set of all gateways of a subview are collected as S_{gateways}. Valid transition region determination is detailed further in Section 3.4.

3.3.2 Waypoint graphs with growing neural gas: Armed with S_{free} alone establishes an operable data set for setting up navigation in any subview S. Many path-planning algorithms could be run directly on this data, though approximation is a fruitful method for improving the performance of path planning. This is the primary goal for utilising waypoint graphs to represent navigable free space. It may be noted that image preprocessing could have also incorporated some dimensionality reduction with simple downscaling instead of operating on full resolutions. However, we have intentionally excluded this to assess the waypoint generation method's approximation with a minimally preprocessed data set.

The GNG algorithm of Fritzke [11] is an ideal fit for learning the topology of navigable space in our two-dimensional subviews. It is an unsupervised learning method for incrementally learning an input data set. From a network, or undirected graph, of a specified number of nodes, or vertices, initialised from a distribution from the input set, the GNG algorithm fits this to the input signal distribution with adjusting size and connectivity of the graph. This fitting is done with adding nodes to the initial network and evaluating deviation error measures of node mappings to input signals to remove nodes and edges contributing to significant error. Training the network with GNG includes many parameters from maximum age of edges, rate of node addition, to error constants. The algorithm inherently includes no stopping criterion though suggestions for maximum graph size or performance measurements are made. A simple instance of this algorithm can be visualised in Fig. 5. This figure shows the GNG graph output at increasing stages of training when applied to an input space represented by the bordering circle.

We directly use the GNG graph approximation as our waypoint graph and formally define it as an undirected graph

$$W = (V_W, E_W).$$

V_W is the set of all waypoint nodes, $\{(x_1, y_1), \ldots, (x_l, y_l)\}$, within each subview S with l being the size of this data set. E_W is the set of all edges that represent a pair of waypoint nodes, with the number of edges being determined in the GNG algorithm. We define our subviews with a size of j data points in Section 3.2,

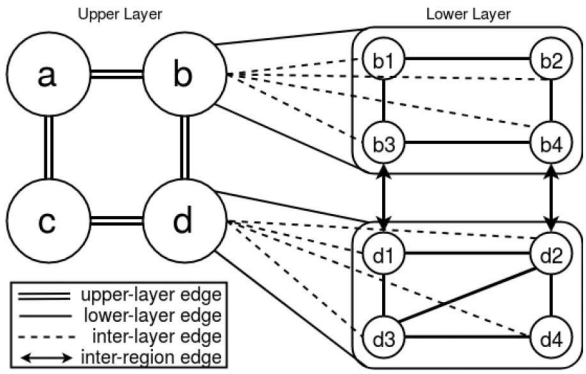

Fig. 6 *Hierarchical graph structure utilised by DISCMAHN*

while our waypoint graph only contains l items. Comparing the size of these two data sets allows us to identify the reduction of complexity that waypoint approximation provides.

It is important that we note the flexibility of the GNG algorithm, as it can adapt to inputs with more dimensions than our current two-dimensional graph requires. Environments do not always have navigable free space that is localised on a singular plane and expanding our subview representation to include a third spatial dimension, depth, would still be appropriate for GNG as a means of generating waypoint graphs.

3.3.3 Graph post-processing: As the previous stage only produces an approximation of our camera's subview, consideration for errors must be made to ensure proceeding stages need not account for errors in our waypoint graph W. Applying GNG to arbitrary topologies could produce approximations with features that may be undesirable and may difficult to target directly. One example of this would be edges of W that intersect any point of $S_{\text{occupancy}}$ that is designated as occupied space. With GNG only operating on the S_{free} data, GNG cannot be made aware of invalid waypoint nodes or edges. Though, as each camera still maintains the state of all points in $S_{\text{occupancy}}$, a blanket solution is to enable pruning of any V_W and E_W if these intersections occur.

3.4 Distributed hierarchical waypoint graph

Up to this point, we have only discussed the environment in terms of subviews, and have only briefly mentioned the relation among them with the definition of our global vision system. Assume our environment, E, as the set of subviews produced by our set of cameras C, where $E = \{S_1, \ldots, S_i\}$. As we intend to evade the constraint of a central server in our system, this understanding of our environment is not useful as no camera or AGV will maintain this representation. It is instead distributed among all cameras as a hierarchical graph such as each waypoint graph, W, local to a camera will be a set of lower-level nodes and each camera will be a higher-level node. That is, each camera will be a parent to a set of nodes that exist within the waypoint graph of its subview. Furthermore, each camera will have siblings that correspond to the neighbouring cameras.

This can be seen in Fig. 6. Each node on the left side with the labels, $\{a, b, c, d\}$, represents one of the overhead smart cameras. These nodes are connected with each of their neighbours and oversee subgraphs. There are two sets of nodes in the lower layer section which represent the region that cameras b and d oversee. Each of the individual nodes in these sets are actual waypoints in the field of view of their respective camera or parent node. There exist inter-region edges on the lower layer between regions overseen by neighbouring upper layer nodes. These are defined by a camera's set of gateways, S_{gateways}, and have weights assigned no cost due to the endpoints overlapping. Furthermore, Fig. 6 shows the relationship between each upper layer node and the nodes within its waypoint graph denoted by the dashed line.

We can define our upper-level system as an indirect graph of parent nodes, $P = (V_P, E_P)$, where V_P is the set of nodes representing the location of cameras in the grid network and E_P defines the set of all edges that represent a pair of neighbouring cameras. Our lower-level system has already been partially defined as it leverages the in-directed waypoint graph, W, of the corresponding parent camera. In terms of our hierarchical grid, we designate $V_{W'}$ as the set of all waypoints and $E_{W'}$ as the set of all waypoint edges over all subviews. With this, we define W' as the set of all waypoint graphs, where

$$W' = (V_{W'}, E_{W'}).$$

Graph P is a parent to graph W', such that the following holds true with a camera node denoted as v_a, a unique subset of waypoints as w, and φ representing a parent–child relation as shown in Fig. 6 by the dashed lines

$$\forall\, v_a \in V_P, \exists\, w \subset W' \mid \exists\, v_a\, \varphi\, w$$

This unique subset of waypoints corresponding to each camera is notable as generating waypoint graphs independently produces a set of disjoint graphs in W', currently leaving no method for path planning to traverse beyond a single subview. As we do not intend to maintain P and W' in a central location, we must ensure that some inter-region connections can be made in a decentralised fashion. Each camera, upon being added to the camera network, will be made aware of P. This upper-level graph is updated to include a new camera node and updates are issued to the rest of the network. With all cameras maintaining a local copy of P, neighbouring cameras can coordinate to establish suitable transition regions or inter-region edges.

This is where the additional steps of image preprocessing to produce S_{gateways} comes in. Using a known neighbour and the arrangement and overlaps of the camera grid P, any camera can utilise its S_{gateways} to find intersections with a neighbouring camera's set of gateways. Specifically, we allow each camera to independently designate transition regions as any subset of S_{gateways} that intersects a border of the parent subview defining a neighbour subview's overlap. We define this intersection as a subview's valid transition region, $T = \{(x_1, y_1), \ldots, (x_m, y_m)\}$, where m is the number of points defining the region. Each camera will collect these regions in $S_{\text{transitions}}$. Due to our grid arrangement and subview overlaps of camera nodes of P, we can ensure that transition region determination is simple and will produce identical regions for topological neighbours of P. Thus, each camera can calculate a centroid of each of its valid transition regions to use as a transition point. Coordination between two neighbouring cameras will synchronise their associated transition points and designate a topological neighbour in P with which this transition point can be used as an inter-region edge in the lower layer of our hierarchical graph. Synchronising transition points allows that each camera can maintain a point guaranteed to be known by a neighbouring camera.

We show a simple example of this with Fig. 7 where we show two disjoint waypoint graphs in W. Fig. 7*a* shows the subview in which each waypoint graph is maintained and shows the overlap of subviews. The left and right subviews both have a constant border width with which S_{gateways} is calculated as shown by the red and blue regions, respectively. The purple region shows the actual overlap of the subviews and thus the intersection of two subview's S_{gateways}. Fig. 7*b* shows one region of the resolved $S_{\text{transitions}}$ set for both subviews in green. Upon coordinating with each other, a centroid of the region is calculated as for the transition point between two subviews shown by the bright green waypoint.

Path planning will utilise these points to create a path between subviews and upon path reconstruction, these points will be disregarded for the nearest waypoint node in relative subviews. We define our set of robots as $R = \{R_1, \ldots, R_n\}$ where the number of robots, n, does not exceed $|V_{W'}|$. The set of robots travel along the nodes of the lower-level graph, W'.

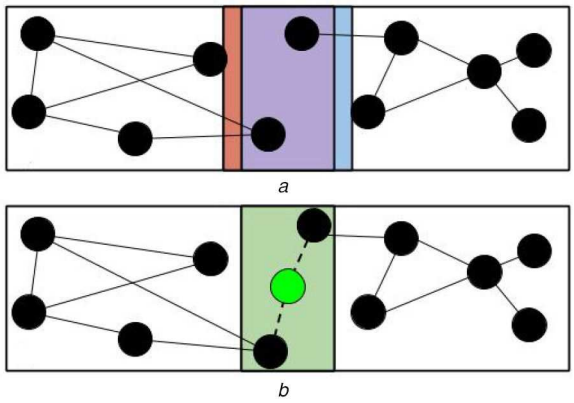

Fig. 7 *Example transition point established to connect two disjoint waypoint graphs*
(a) Disjoint waypoint graphs preceding insertion of transition point, ***(b)*** The resulting waypoint graph after insertion of transition point

Require:
Current Camera, C_C
Neighbor Camera, C_N
if C_C is *this* **then**
  Let P_O represent a point in the overlap region between C_C and C_N
  return path_cost(C_C.startPoint, P_O)
else
  return C_C.request_cost(C_C, C_N)
end if

Fig. 8 *Algorithm 1: D-A* cost*

Require:
Goal Camera, C_G
Goal Point, P_G in C_G
Let $C = C_G$
Let P_F be an initially empty set of edges representing the final path
Let P_T be an initially empty set of edges representing an intermediate path
while C.parent $\neq$ NULL **do**
  P_T = *this*.requestPath(C.startPoint, C.endPoint)
  P_T.append(P_F)
  $P_F = P_T$
  P_T.clear()
end while

Fig. 9 *Algorithm 2: rebuild path*

3.5 Path planning

To the authors' knowledge, a method of path planning across a decentralised system we have defined does not exist. Similar work has been done for centralised systems in [8, 9]. Therefore, we propose a modification to standard A*. As we do not have another method to compare against, we will use a greedy Euclidean-based search as a comparison. We are assuming that a camera in the upper layer knows of other cameras' positions relative to itself.

3.5.1 Greedy search: For the greedy method of path planning, we only take into account the Euclidean distance between the current camera and the goal camera. Therefore, nothing at the child level of the graph is considered. At each camera, we take the neighbour with the lower Euclidean distance to the goal camera and set it as the current camera. This would be equivalent to finding the shortest path through a camera in the direction of the goal and re-planning at each camera. This method runs until the goal is found. The path is then rebuilt as described in Algorithm 2.

3.5.2 Decentralised-A:* We propose Decentralised A* (D-A*), a modified version of A* to plan a path across a set of decentralised, connected parent nodes. The children of these nodes will be taken into account during the planning process. Work in decentralised path planning has been done in [17, 18]. However, these algorithms run in a decentralised system where a robot or set of robots is self-sufficient. In DISCMAHN, we have a set of parent nodes creating a path through their children which the robots will follow. As it is

Require:
Start Camera, C_s
Goal Camera, C_g
Start Point P_s in C_s
1: add C_s to *openSet*
2: **while** *openSet* is not empty **do**
3: N_C = *openSet.pop*
4: **if** N_C is C_g **then**
5: **return** C_S.rebuild_path()
6: **end if**
7: **for** each *neighbor* of N_C **do**
8: **if** *neighbor* in *closedSet* **then**
9: **continue**
10: **end if**
11: **if** *neighbor* not in *openSet* **then**
12: add *neighbor* to *openSet*
13: **end if**
14: *potentialCost* = N_C.costSoFar + c_s.cost(N_C, *neighbor*)
15: **if** *neighbor* is C_G **then**
16: *ov_point* = find_overlap_point(N_C, C_g
17: *potentialCost* = C_S.getPathCost(C_G.startPoint, C_G.endPoint)
18: **end if**
19: **if** *potential_cost* < *neighbor*.costSoFar **then**
20: N_C.endPoint = find_overlap_point(N_C, neighbor)
21: neighbor.startPoint = N_C.endPoint
22: *neighbor*.parent = N_C
23: *neighbor*.costSoFar = *potentialCost*
24: *neighbor*.costToEnd ← *neighbor*.costSoFar + heuristic()
25: **end if**
26: **end for**
27: *closedSet*.add(N_C)
28: *openSet*.remove(N_C)
29: **end while**
30: **return** failure

Fig. 10 *Algorithm 3: D-A**

decentralised, each parent node will not know of the internals of its neighbours. Therefore, it will be necessary to use some form of network communication to request necessary information from other parent nodes. This algorithm follows a top-down path-finding approach. The majority of the path finding will happen on the parent layer with information such as cost and heuristic coming from the child layer of the requested camera. The cost will be the shortest path across the child layer from the starting edge of the current camera to a point in the overlap region of the neighbouring camera. For this algorithm, it is assumed that a path-finding algorithm to determine the shortest paths on the child layer of every camera has already run. The primary difference between this and standard A* is the cost function and path rebuilding. This algorithm runs solely on the source camera which forces communication with other cameras for information. The path will be entirely rebuilt on the source camera and passed along to the robot. Given all of this, we propose the following modifications to standard cost, path reconstruction, and A* algorithms in Algorithms 1, 2, and 3 respectively (see Figs. 8–10).

As per Algorithm 1 (Fig. 8), if the source node is not the node we are seeking cost for, it will request cost from C_C for a path from the start point of C_C to a point in the overlap region shared with the neighbour in question. The cost of the path currently consists only of the length. After the goal is found, the path will be rebuilt starting from the goal camera as described in Algorithm 2 (Fig. 9). The source camera will request the path between the start and end points from the goal camera, and append it to the final path. The interior of the loop concatenates the most recently requested path onto the front of the final path. Since we are operating on a standard grid, the currently used heuristic is Euclidean distance. However, because this is a modification to standard A*, the heuristic can be changed to fit the situation. The final modification to A* is on line 15 of Algorithm 3 (Fig. 10). The cost in this implementation is directional dependent; if the current node is the immediate neighbour of the goal camera, the cost of the path to the final goal point must be considered. Assume a goal camera, C_G has two neighbours, N_1 and N_2. If cost(N_1, C_G) < cost(N_2, C_G), then A* will assign N_1 as the parent of C_G. Assume O_{P1} is the overlap point between N_1 and C_G. Assume O_{P2} is the overlap point between N_2

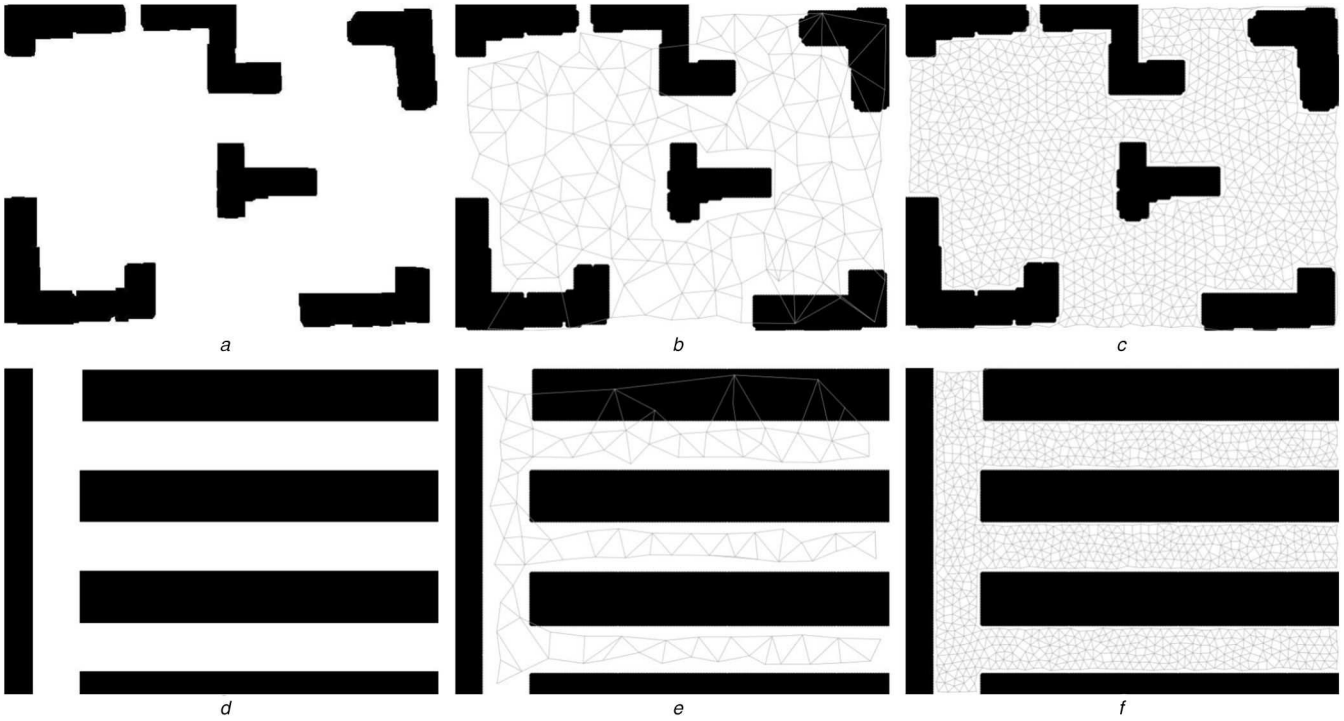

Fig. 11 *Waypoint generation training on subviews*
(a)–(c) Waypoint generation training on a lab captured environment, with the original $S_{\text{occupancy}}$ image in a, and the resulting waypoint graph, W, after 1 (b) and 20 (c) training epochs, *(d)–(f)* Identical stages, though for a subview of our generated test environment

and C_G. Due to cost being directional dependent, this could cause a problem when finding the path from the start point of C_G to the goal point. This is because it is possible that

$$\text{pathCost}(O_{P2}, \text{goalPoint}) < \text{pathCost}(O_{P1}, \text{goalPoint})$$

If the difference is large enough, it may be true that the cost required to get to the goal point would be lower if we followed the path along N_2

$$\text{cost}(N_1, C_G) + \text{pathCost}(O_{P1}, C_G.\text{endpoint}) >$$

$$\text{cost}(N_2, C_G) + \text{pathCost}(O_{P2}, C_G.\text{endpoint})$$

This would mean the path going through N_1 is sub-optimal. Therefore, we must take the extra path cost into account when assigning a parent to the goal node.

4 Evaluation

We direct our evaluation towards the waypoint generation and path planning aspects of DISCMAHN. We tested waypoint generation on an array of images captured in our lab where we attempted to replicate an industrial warehouse storage facility that was scaled down with one capture previously shown in Fig. 4. Additionally, we generated a much larger environment, also roughly resembling an industrial warehouse, to ease the evaluation of waypoint generation as well as transition region determination for a grid of subviews. Path-planning simulations were performed to easily compare our D-A* to a standard method.

4.1 Waypoint generation

As we rely on the GNG algorithm for producing our waypoint graphs, we sought an implementation for our evaluations. After stumbling upon the work of Shevchuk [19], which discusses the use of GNG for making aesthetic changes to images with the use of GNG, the library used by this work was further explored. The author was revealed to have also created the NeuPy library for Python. In short, this toolkit allows the exploration of neural networks and enables a specific GNG algorithm class. This class was directly implemented from the work of Fritzke [11] and allows extracting a graph, though it includes two modifications not presented there. With the possibility of increasing the number of start nodes and speeding up the training step with fewer graph updates [20], we chose to evaluate the waypoint generation of our DISCMAHN system with Python.

We tested waypoint generation on various topologies with one test on an actual captured image, and the rest on a generated environment. As we are referring to waypoint generation here, it is important to note that we operate on our binary images, $S_{\text{occupancy}}$, achieved from image preprocessing. For our captured image test, we use the process of [16] to create $S_{\text{occupancy}}$, while our generated binary environment's subviews are an inherent representation of $S_{\text{occupancy}}$, themselves. Our captured image, in Figs. 11*a–c*, is a resolution of 640 × 480 pixels. We aimed for realistic resolutions to match captured images with our generated environment and thus created an image suitable for splitting into equivalent subviews. This environment, seen in Fig. 12, is a 3 × 3 grid of subviews, each with a 640 × 480 resolution, for a total resolution of 1880 × 1440 pixels. This also includes a 20-pixel overlap of all neighbouring cameras subviews. We show one subview of this environment in Figs. 11*d–f*.

We utilised a simple python script for training a GNG instance with NeuPy for every subview. There are numerous parameters for GNG as well as those introduced with NeuPy's implementation. We do no tackle this optimisation problem here and conversely just define the major parameter values we assigned across all tests that differ from default in the NeuPy GNG class. We begin all networks with two nodes, permit a maximum network size of 1000 nodes, and force new nodes every 100 iterations of the algorithm.

Figs. 11*a–c* show our captured lab environment in its original form, at epoch 1, and epoch 20 of training, respectively. Figs. 11*d–f* show a subview of our generated test environment in its original form, at epoch 1, and epoch 20 of training, respectively. It is evident that this algorithm can produce accurate approximations of unoccupied space in a subview with sufficient training. However, it is difficult to define sufficiently, so we have established 20 epochs of training to be sufficient for our demonstration purposes. As can be seen in Figs. 11*b* and *e*, insufficient training can lead to a poor approximation of navigable space in addition to producing invalid waypoints and connecting edges that intersect occupied space. Our

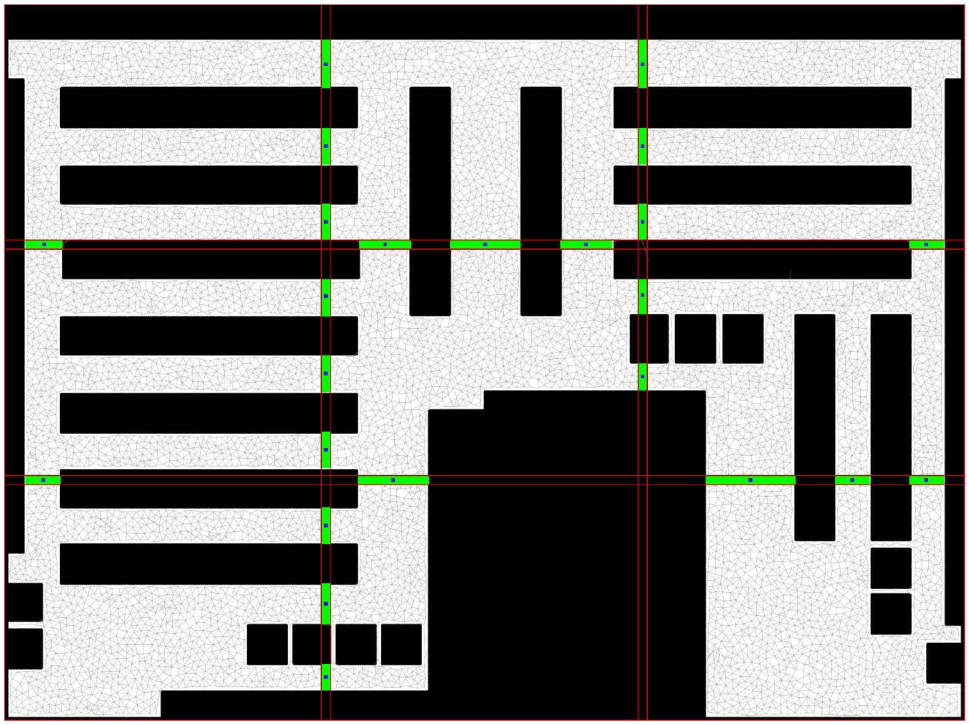

Fig. 12 *Generated test environment with calculated waypoint graphs, transition regions, and transition points*

proposed post-processing of the waypoint graph is essential to remove these errors.

In the embodiment of waypoint generation as described in this work, we are approximating free space of single images. As DISCMAHN is intended to be resilient to dynamic environments, the waypoint generation process must be analysed in terms of runtime. To do so, we have downscaled input images to 25% of their original size and fed the S_{free} set to the NeuPy GNG implementation. Using the above-aforementioned parameters, we show the runtime and node count of every other 20 epochs of training for our lab captured image and three individual subviews, subviews 1, 7, and 8, of our generated test environment. These can be seen in Tables 1 and 2. The subviews of Fig. 12 are indexed 0–8 beginning from the top left corner and proceeding by rows followed by columns in the subview grid. These NeuPy GNG training runtimes were collected on Mac OSX 10.15 with a 3.7 GHz Quad-Core Intel Xeon E5.

Table 1 Waypoint generation training runtimes

Epoch	Lab image Runtime, s	Subview 1 Runtime, s	Subview 7 Runtime, s	Subview 8 Runtime, s
2	4.2232	2.9965	0.6575	3.0472
4	7.5386	5.2090	0.9449	4.9890
6	10.6436	7.2900	1.2386	7.2372
8	14.4911	11.0429	1.5104	9.6058
10	13.6929	10.6668	1.6893	10.4190
12	12.9513	10.2359	2.0163	10.2781
14	13.0099	10.5054	2.4023	10.7294
16	13.7486	10.4434	2.4934	12.0525
18	13.2121	10.3286	2.8863	10.7865
20	12.7866	10.4412	3.0714	10.3710

Table 2 Waypoint generation training node counts

Epoch	Lab image No. Waypoints	Subview 1 No. Waypoints	Subview 7 No. Waypoints	Subview 8 No. Waypoints
2	295	242	84	240
4	589	483	166	478
6	882	724	249	716
8	1000	965	331	954
10	1000	1000	413	1000
12	1000	1000	496	1000
14	1000	1000	578	1000
16	1000	1000	660	1000
18	1000	1000	743	1000
20	1000	1000	825	1000

With the data in Table 1, it is evident that the time to generate a suitable waypoint graph for a subview is unfeasible for real-time applications. This runtime is feasible only for the scenario in which an initial map is generated. Subsequent updates to a waypoint graph for every new input image could enable real-time maintenance of a waypoint graph, though this is a topic of on-going research. However, a few things are notable with these results. We see the waypoint graph become fully saturated rather early on in training with our lab image and two of our subviews. Continued training on a saturated network significantly slows down the network learning rate and thus tuning the maximum node count could be beneficial, here. Additionally, it can be noted that the subviews with saturated waypoint graphs contain more free space than subview 7 does. This suggests that identifying a proper maximum waypoint count could be done as a function of the size of free space. Runtimes can be seen to be directly influenced by increasing waypoint node counts though more so are influenced by the size of input data. This data suggests that additional considerations will need to be paid to downscaling data sets with future implementations aimed for real-time waypoint graph updates.

To demonstrate the transition region determination, we include Fig. 12. This shows the set $S_{\text{transitions}}$ for each subview in green with the centroid transition point that is used to create an inter-region edge between the disjoint waypoint graphs of each subview. The red borders dictate the subview boundaries.

4.2 Path planning

To test the effectiveness of path planning, we simulated an environment of varying sizes. For the top layer, we created a layer of size 5×5, 10×10, 20×20, and 50×50 parent nodes. Each parent node in the layer was given a random number of children between 15 and 20. Random edges were generated among the children of each parent node with the chance of an edge forming

Table 3 Path lengths normalised to greedy

Grid dimensions	25% node connectivity	50% node connectivity	75% node connectivity
5 × 5	0.888625	0.902511	0.954489
10 × 10	0.854601	0.871142	0.894761
20 × 20	0.837341	0.856450	0.880150
50 × 50	0.810573	0.848751	0.855922

Table 4 Request counts normalised to greedy

Grid dimensions	25% node connectivity	50% node connectivity	75% node connectivity
5 × 5	3.59	3.77	3.28
10 × 10	6.04	5.72	5.53
20 × 20	9.67	9.77	9.76
50 × 50	22.58	22.08	22.43

Table 5 Runtimes normalised to greedy

Grid dimensions	25% node connectivity	50% node connectivity	75% node connectivity
5 × 5	13.145036	13.174933	13.219170
10 × 10	23.257459	21.718941	21.236186
20 × 20	35.641699	37.617155	38.646010
50 × 50	83.810102	82.483988	83.024822

between any two child nodes being 25, 50, and 75%. This is the connectivity of the graph. In each test run, both pathing algorithms were run on the same graph from the same start point to the same goal point. All of the values in Tables 3–5 represent the data collected from D-A* normalised by the values collected from greedy. These values represent the normalised mean for over 100 tests for each combination of the above grid sizes and connectivity rates.

In Table 3, it can be seen that as the connectivity rate increases, the benefit of using D-A* diminishes. This is expected because as the graph becomes closer to fully connected, greedy will be more likely to find an optimal path. It can also be seen that as the size of the grid increases, the benefit of using D-A* increases. This is also expected because as the size of the graph increases, it is more likely that a shorter, less direct path exists. Table 4 shows that connectivity has no real effect on the number of requests needed to create a path across a constant grid size. This is because the connectivity rate is local to each parent node's child graph. Therefore, even if every child was fully connected, the parent layer, where communication takes place, connectivity does not change. On the other hand, the number of requests greatly increases as the size of the parent layer increases. This is because D-A* will have a higher branching factor and greater depth as the size of the graph being traversed increases, requiring more communication. The results in Table 5 follow the same pattern as Table 4 for the same reasons. The connectivity of the child layer does not affect the efficiency of D-A* running on the parent layer. The runtime also increases greatly with the size of the parent layer due to having a larger search space. It should be noted that while D-A* has a very high runtime relative to greedy search, the actual runtime was still under one-tenth of a second on the 50 × 50 layer on average.

5 Conclusion

In this work, we presented DISCMAHN, a system designed to work in a dynamic environment without loss of efficiency in navigation. We present a process by which waypoint graphs can be generated from an image from an overhead smart camera. A distributed hierarchical graph allows the global vision system, enabled by our overhead cameras, to be easily navigated as dictated by the camera network. Furthermore, we present D-A*, a modification of the standard A* algorithm which allows decentralised path-planning to be done. Previous methods discussed robot navigation for static and dynamic environments which require remapping of the environment when changes are made to the environment. Using DISCMAHN, waypoints can be generated anytime a change is detected within the environment from the overhead smart cameras, though speed up is necessary. Once all waypoints are generated, D-A* runs on the monitoring camera to plan a decentralised path between any two locations within the environment. Allocating a single camera to handle path planning through over multiple fields of view avoids the need for new paths upon entering a new subview.

For future works, the authors will expand path planning to include instances of deadlock. Also, we will consider the implementation of a fail-safe system on the robots. The authors do believe that these considerations with optimising our waypoint generation for real-time applications as well as our D-A* algorithm will create a more efficient unmanned robotic coordination system.

6 References

[1] Parker, L.E.: 'Path planning and motion coordination in multiple mobile robot teams', *Encycl. Complexity Syst. Sci.*, 2009, pp. 5783–5800

[2] Dissanayake, M.G., Newman, P., Clark, S., *et al.*: 'A solution to the simultaneous localization and map building (SLAM) problem', *IEEE Trans. Robot. Autom.*, 2001, **17**, (3), pp. 229–241

[3] Olson, E.: 'Apriltag: A robust and flexible visual fiducial system'. 2011 IEEE Int. Conf. on Robotics and Automation, Shanghai, 2011, pp. 3400–3407

[4] Johnson, E., Olson, E., Boonthum-Denecke, C.: 'Robot localization using overhead camera and leds'. Twenty-Fifth Int. FLAIRS Conf., Marco Island, Florida, USA, 2012

[5] Ball, D.M., Wyeth, G.F., Nuske, S.: 'A global vision system for a robot soccer team'. The 2004 Australasian Conf. on Robotics and Automation (ACRA 2004), Canberry, Australia, 2004

[6] Brezak, M., Petrović, I., Ivanjko, E.: 'Robust and accurate global vision system for real time tracking of multiple mobile robots', *Robot. Auton. Syst.*, 2008, **56**, pp. 213–230

[7] Hart, P.E., Nilsson, N.J., Raphael, B.: 'A formal basis for the heuristic determination of Minimum cost paths', *IEEE Trans. Syst. Sci. Cybern.*, 1968, **4**, pp. 100–107

[8] Zhong, C., Liu, S., Zhang, B., *et al.*: 'A fast on-line global path planning algorithm based on regionalized roadmap for robot navigation', *IFAC-PapersOnLine*, 2017, **50**, (1), pp. 319–324

[9] Lee, J.Y., Yu, W.: 'A coarse-to-fine approach for fast path finding for mobile robots'. Proc. from IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, St Louis, St Louis, United States, 2009, pp. 5414–5419

[10] Felder, A., VanBuskirk, D., Bobda, C.: 'Automatic generation of waypoint graphs from distributed ceiling-mounted smart cameras for decentralized multi-robot indoor navigation'. Proc. of the 13th Int. Conf. on Distributed Smart Cameras (ICDSC 2019), Via Vigilio Inama, 2019, 5, 38122 Trento TN

[11] Fritzke, B.: 'A growing neural gas network learns topologies', *Advances in neural information processing systems*, 1995, pp. 625–632

[12] Tence, F., Gaubert, L., Soler, J., *et al.*: 'Stable growing neural gas: a topology learning algorithm based on player tracking in video games', *Appl. Soft Comput.*, 2013, **13**, (10), pp. 4174–4184

[13] Dellinger, B., Jenkins, R., Walton, J.: 'Automated waypoint generation with the growing neural gas algorithm'. Florida Artificial Intelligence Research Society Conf., May 2017

[14] Quigley, M., Conley, K., Gerkey, B.P., *et al.*: 'ROS: an open-source robot operating system'. ICRA Workshop on Open Source Software, Kobe, Japan, 2009

[15] Comaniciu, D., Meer, P.: 'Mean shift: a robust approach toward feature space analysis', *IEEE Trans. Pattern Anal. Mach. Intell.*, 2002, **24**, (5), pp. 24–24

[16] Streit, F.J., Pantho, M.J.H., Bobda, C., *et al.*: 'Vision-based path construction and maintenance for indoor guidance of autonomous ground vehicles based on collaborative smart cameras'. Proc. of the 10th Int. Conf. on Distributed Smart Camera, Paris, France, 2016, pp. 44–49

[17] Velagapudi, P., Sycara, K., Scerri, P.: 'Decentralized prioritized planning in large multirobot teams'. 2010 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, Taipei, Taiwan, 2010, pp. 4603–4609

[18] Draganjac, I., Miklić, D., Kovačić, Z., *et al.*: 'Decentralized control of multi-AGV systems in autonomous warehousing applications', *Proc. IEEE Trans. Autom. Sci. Eng.*, 2016, **13**, (4), pp. 1433–1447

[19] Shevchuk, Y.: 'Making art with growing neural gas', NeuPy.com, March 2018

[20] Shevchuk, Y.: 'Growing neural gas (GNG) algorithm', NeuPy.com, March 2018