Robotica (2017) volume 35, pp. 1280–1309. © Cambridge University Press 2016
doi:10.1017/S0263574716000047

An approach for real-time motion planning of an inchworm robot in complex steel bridge environments

David Pagano* and Dikai Liu

Centre for Autonomous Systems, University of Technology Sydney Broadway, NSW 2007, Australia.
E-mail: dikai.liu@uts.edu.au

(Accepted January 13, 2016. First published online: February 11, 2016)

SUMMARY
Path planning can be difficult and time consuming for inchworm robots especially when operating in complex 3D environments such as steel bridges. Confined areas may prevent a robot from extensively searching the environment by limiting its mobility. An approach for real-time path planning is presented. This approach first uses the concept of line-of-sight (LoS) to find waypoints from the start pose to the end node. It then plans smooth, collision-free motion for a robot to move between waypoints using a 3D-F^2 algorithm. Extensive simulations and experiments are conducted in 2D and 3D scenarios to verify the approach.

KEYWORDS: Inchworm robot; Motion planning; Complex environments; Real-time; Line-of-sight.

1. Introduction

Many steel structures require continuous maintenance and inspection to ensure longevity, structural integrity and aesthetics. Maintenance and inspection procedures are highly demanding especially when the structure is used extensively by trains, cars and pedestrians, or located in destructive surroundings such as marine environments. Maintenance teams are often required, not only to perform the inspection and maintenance, but to perform support roles such as monitoring equipment and emergency response. Hazardous conditions due to heights, confined spaces or external environmental factors can compound the difficulty in performing maintenance and inspection procedures.[27,36]

Deploying a climbing robot eliminates many of the issues associated with manual maintenance and inspection. By using a robot, human exposure to hazardous conditions is reduced, which also reduces the number of support personnel required. Additionally, the robot is able to inspect and navigate areas that a human would not be able to access without support equipment or personal protective equipment to protect them from dangerous elements such as excessive heat, fumes or contaminants.

Many climbing robots exist which are capable of inspecting a wide variety of structures. CLIBO[40] uses claws to climb rough, vertical surfaces using motion techniques that both mimic rock climbers and cats. W-Climbot[8] optimises the surface placement of its suction caps on climbing surfaces through pose selection techniques to improve the efficiency of the vacuum. Waalbot II[26] utilises micro-fibre footpads to attach to a number of different surfaces. By using a number of external sensors, Wallbot II is able to plan trajectories, using Euclidean distance heuristics and A* searches, across connected regions of the environment to maximise adhesion and decrease the chances of catastrophic failure. Robots have different adhesion methods, and use different control techniques, appropriate to their application and to overcome environmental constraints they may encounter.[37] Steel bridges and other complex ferromagnetic structures lend themselves to being climbed by inchworm robots, as the many degrees of freedom (DOF) of the body allow transitions between surfaces with high angular variation and the ability to navigate around smaller obstacles such as rivets, with magnetic pads for adhesion to the ferromagnetic surface.[30]

* Corresponding author. E-mail: david.pagano@student.uts.edu.au

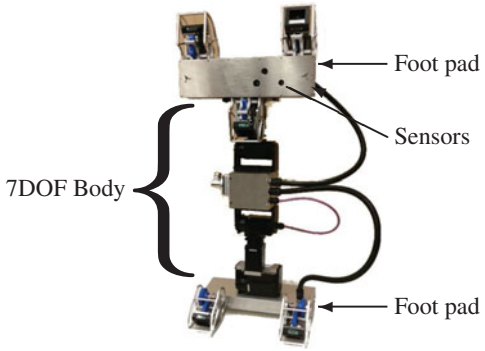


Fig. 1. A 7DOF inchworm robot for steel bridge inspection.

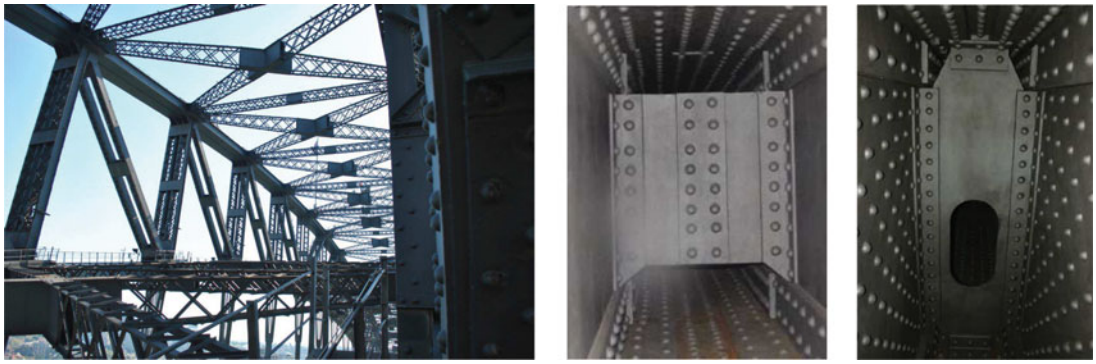


Fig. 2. (a) A complex bridge scenario and related difficult areas; (b) a partition plate with limited space on the top and bottom for access and (c) a partition plate with a small manhole for access.

An inchworm robot[43] (Fig. 1) has been designed for the application scenario of inspection of a steel bridge (Fig. 2) focusing on areas which are particularly difficult and hazardous for personnel to access. While navigating the exterior of the bridge is difficult, navigating the interior presents a number of additional challenges. Limited access prevents acquisition of *a priori* knowledge; confined spaces and small gaps limit the size of the robot; riveted surfaces and poor surface condition reduces the available surfaces for reliable attachment. Two especially difficult and hazardous areas occur within the tunnel substructure of the bridge; a partition plate with limited access through the top and bottom, and a partition plate with a manhole through the centre (Figs. 2b and 2c, respectively). The inchworm robot provides a suitable platform for navigation through each partition as it is capable of stepping between rivets while maintaining the necessary manoeuvrability to bypass the partitions.

However, controlling robots, such as inchworm robots, in complex 3D environments is difficult. Many motion planners, particularly offline planners, require information of the environment to find a complete solution. Provided the environment is known, a solution, albeit potentially a sub-optimal one, will be found. If the optimal solution is required then the entire environment must be mapped *a priori*. In reality, planners may need to function on partially known environments or be required to update the path reactively to new information. Online planners capable of using real-time sensor data become preferential in unknown environments where optimal global solutions are not feasible due to limited information.[4,38] However, some online planners do not guarantee solutions, let alone optimal solutions.

In general, autonomous robots can be slow in performing many actions, such as searching, mapping and exploration. By improving the efficiency of algorithms, robots become more practically applicable. One avenue of improving efficiency is by taking inspiration from techniques humans intrinsically use to perform tasks. This paper presents an approach to find a path for an inchworm robot around complex environments and is comprised of two algorithms: a Line-of-Sight Tree (LoST)

https://doi.org/10.1017/S0263574716000047 Published online by Cambridge University Press

algorithm, which is used to determine waypoints, and a 3D force field (3D-F^2) algorithm[5] which develops smooth, collision-free paths between waypoints.

The approach draws inspiration loosely from how humans perceive a scene whereby their gaze tends to fixate on important and information-rich regions, such as contrasting colours or the edges of objects.[15] These regions subsequently direct their gaze to other regions. Our approach similarly uses this concept to find the end node. A node is described as a possible robot position in Cartesian space; $\{x, y\}$ in 2D and $\{x, y, z\}$ in 3D. The end node describes the desired final robot position. The approach directs the inchworm robot to "look" around its initial position for the end node using a Microsoft Kinect sensor. If the end node is not found, the LoST algorithm finds obstacle edges, within LoS of its current position, to place intermediate nodes beyond. The visible region refers to the visibility of the area surrounding a point in the environment; while the LoS is the unimpeded view between two points within the environment. The 3D-F^2 algorithm then finds a smooth, collision-free path for the inchworm robot towards the selected node. This process continues until the end node is found.

The LoST algorithm determines visible regions by ray casting omnidirectionally from a node. Ray casting provides a quick and computationally efficient method to determine the distance from the node to environmental obstacles.[1] As ray casting does not need previous environmental knowledge, the LoST algorithm can subsequently be suitable for online applications in unknown environments. While ray casting is used in this algorithm, this method is interchangeable with any sensors or methods which determine the visible regions within a set field of view.

Our approach allows for symbiosis between the LoST and 3D-F^2 algorithms. The LoST algorithm develops obstacle free paths, i.e. way points, which cannot be directly usable for smooth motion of an inchworm robot. The 3D-F^2 algorithm can generate smooth, collision-free motion, but it may become trapped in local minima; regions where the sum of attractive and repulsive forces are zero, stalling further motion.[22] As intermediate nodes found by the LoST algorithm are found through LoS, they are unimpeded by obstacles. Similarly, the attractive force generated by the 3D-F^2 algorithm will be unimpeded as no obstacles will be present between two adjacent intermediate nodes. This reduces the chances of generating repulsive forces directly opposing the motion and entering into a local minima.

In Section 2, biologically inspired techniques related to motion planning are discussed. Section 3 focuses on the methodology behind the approach and details of the LoST and 3D-F^2 algorithms. Experimental results and discussion are presented in Section 4. Section 5 describes some drawbacks with the LoST algorithm. In Section 6, conclusions on the presented approach will be given.

2. Related Work

Many motion planners are capable of planning robot motion in both known and unknown environments. While offline planners are capable of finding paths if a significant amount of the environment is known, online planners are preferred in unknown environments. The solutions online planners find can potentially be suboptimal and generally take longer times to find as knowledge needs to be gathered in real-time through sensor information. Improvements to the efficiency of motion planning algorithms will reduce the overall computational effort of robotic systems allowing them to become more practically applicable.

2.1. Bio-inspired planning algorithms

Many online planners are capable of finding paths through varied environments. By improving algorithm and software efficiency, robots may be utilised in more complex environments. Ideas for improvements to efficiency can be inspired by the actions undertaken by humans and animals.[23]

Particle swarm optimisation (PSO)[20] was developed based on the interpretation of the movements of flocks of birds or schools of fish. The underlying rules which enabled large groups of organisms to synchronise after sudden changes in motion were modelled. This amounted to a nearest neighbour velocity matching with a factor, dubbed "craziness", added to prevent a unanimous, unchanging model. Further development of the model allows PSO to optimise a wide variety of non-linear systems. PSO was used to determine the parameters of a virtual force field algorithm (VFF) which would successfully allow navigation of four robots in various environments.[42] A variation of PSO, a phase angle-encoded and quantum-behaved PSO, was developed for control of an unmanned aerial vehicle.[11] Tests showed that this variation provided improved performance over similar algorithms.

https://doi.org/10.1017/S0263574716000047 Published online by Cambridge University Press

The behaviour animals exhibit when interacting with pheromones has been applied to mobile robotics. An algorithm is presented for searching through an environment using small robots dispersed throughout.[33] The first robot to find the target emits a virtual pheromone. This virtual pheromone is detected by nearby robots which have a direct, uninterrupted path to the first robot. These robots similarly emit virtual pheromones and the process continues until the origin is reached. Following the pheromone trail from the origin back through each robot to the first robot reveals the path to the target. Pheromones are also used in Ant Colony Optimisation algorithms[7] whereby paths are found based on virtual ant pheromone levels. Initially a single "ant" is released to randomly search for a target in a seeded environment. As it travels between seeds, it leaves a pheromone trail. The shorter the distance travelled the higher the level of the pheromone trail left behind. Once the target is found, more ants are released to move towards the goal in a probabilistic manner based on heuristics and the pheromones left by preceding ants. As more ants reach the goal, the overall pheromone level is reduced. The overall pheromone level continues to reduce until converging upon the optimal solution.[3]

A control algorithm was developed for a manipulator with tactile sensors based on the principle that animals embrace contact with obstacles rather than avoiding it altogether.[16] While limited to a single plane of motion, the manipulator is moved towards a goal within a cluttered environment, with contact forces generated when contact is made with obstacles. These contact forces are used to determine repulsive forces which act on the manipulator to prevent excessive collisions and potential damage. An emerging trend in robotics is the concept of socially acceptable collision avoidance[39] whereby, when avoiding pedestrians, a robot mimics "social norms" preventing abrupt, and potentially dangerous, motions from pedestrians. Here, a variation of the Social Force Model is applied to a robot controller to maintain a social distance from pedestrians. This resulted in safer navigation and no unsafe motions when among pedestrians.

2.2. Motion planning algorithms

Potential field (PF) based planning algorithms[21] have been in development for years. From the start position at a high potential, a path is generated which tends away from high potentials regions around obstacles while moving towards the goal position at the low potential. A variation of the PF was developed called the VFF algorithm.[2] Instead of evaluating the entire map as PF algorithms do, VFF algorithms use sensor data to determine repulsive force based on the distance to obstacles directly surrounding the robot. However, basic PF and VFF planners may encounter regions known as local minima. This occurs when the repulsive forces directly oppose the attractive force resulting in a zero sum and stalling further motion.[22] Unless avoidance techniques are implemented, it is difficult for robots to escape local minima.[24]

A 3D-F^2 algorithm, a variation of the VFF algorithm, is used in conjunction with a haptic device and a Human–Machine-Interface to teleoperate a Denso robot.[5] An operator moves the robot by controlling the attractive forces acting on the end-effector via a 3DOF haptic device while the algorithm calculates repulsive forces to avoid obstacles. The forces are then applied to the robot's dynamics to determine joint motion. Chotiprayanakul successfully implemented this algorithm and tested it in field trails; however, he acknowledges more evaluations are necessary. As Chotiprayanakul implementation takes advantage of a human operator, no local minima avoidance technique is implemented. The concept of depth space[10] is introduced for controlling a 7DOF manipulator in dynamic environments. An external depth sensor is used to evaluate the distances between known dynamic obstacles and the manipulator, projecting this information onto a $2\,^1/_2$ dimensional depth space. Simplified distance calculations are used to quickly gather the minimum information required to generate repulsive forces for a PF algorithm to perform obstacle avoidance. These forces are generated so as to allow the manipulator to pivot away from obstacles moving towards it faster than it could otherwise. An extension of the PF algorithm[25] is used to navigate in cluttered and unknown 2D environments. A robot is driven towards a goal point until an obstacle is reached, upon which it navigates towards the closest gap while maintaining parallel motion along the obstacle. Once the obstacle is bypassed, the robot continues towards the goal. Results showed improved robot speeds and less oscillatory motions when compared to similar algorithms.

Rapidly-exploring random tree (RRT) algorithms offer a simple method for effectively searching known environments. However, they intrinsically have difficulties finding paths through narrow gaps in the environment. A method to overcome this is to establish a new tree at nodes which cannot be connected to an existing tree.[6] This variation has improved convergence rates over the standard RRT algorithm. Online RRT algorithms are capable of replanning sections of the current tree based

on new sensor information or dynamic obstacles.[9] Invalid sections are removed with the affected branches reassessed and regrown until a successful solution is found again. RRT* algorithms[19] find asymptotically optimal solutions by finding optimal paths to every state in the problem domain. As this can be expensive in high-dimensional state spaces, an Informed RRT* algorithm[12] limits the search space once an initial solution has been found to within a prolate hyperspheroid encompassing the initial and goal states. By focusing the search space, the Informed RRT* algorithm approaches optimal solutions sooner in comparison to regular RRT* algorithms. While these algorithms function well in both 2D and 3D, functionality is limited to relatively simple dynamic models.

A simultaneous path planning and mapping algorithm[13] is presented which incrementally constructs a map using local sensor data. Scans are taken and a Hierarchical Topology Map is created based on admissible free space. Results proved successful. However, the time taken was dependent on the sensor capabilities in relation to the map size and robot motion could be slow in highly cluttered environments. A visibility binary tree algorithm[35] is presented based on the visible tangents between a robot and obstacles. The visibility binary tree is built by first simplifying the robot and obstacles to circular shapes. Then, from the robot's start position, a direct path to the target is sought. If a collision occurs, the path is split tangentially in two around the obstacle. This process is then repeated with each new vertex and continues until all vertices are connected to the target. The final tree is optimised to remove redundant edges and searched through to find the shortest path. Results showed good performance in both known and unknown environments with better optimised paths than compared algorithms. The Dynamic Brushfire algorithm is developed to update generalised Voroni diagrams in dynamically changing environments.[18] As new information is usually localised around the robot, replanning the entire environment is unnecessary. Instead the Dynamic Brushfire algorithm only replans the affected portion of the map, a functionality previous generalised Voroni diagrams lacked, increasing the overall efficiency of the algorithm. These algorithms proved successful for collision avoidance using graph-based searches in 2D environments.

2.3. Hybrid motion planning algorithms

Hybrid motion planning algorithms often provide additional benefits in comparison to conventional planning algorithms. They can provide a greater chance of finding a successful path, generate smarter paths or overcome specific problems such as local minima.

By combining an RRT algorithm with an A* algorithm, a planning algorithm is presented which is capable of finding paths to multiple, unknown goal poses.[41] The A* algorithm is used to find a path to satisfy the given problem requirement. The space is then explored further, using the RRT algorithm, to find additional paths which satisfy the problem requirement. Similar paths are then clustered together. This combined algorithm was run on a number of environments with results demonstrating improved performance when compared to individual A* and RRT algorithms. A potentially guided directional-RRT* algorithm[34] improves the efficiency of RRT* algorithms by directing generated trees towards regions of decreasing potential within a known environment. Results showed a reduction in the number of iterations and in the time required to compute optimal solutions in comparison with RRT* algorithms. However, both hybrid algorithms consider only simple kinematic models.

A hybrid circular fields and PF algorithm was developed for use in dynamic and partially unknown environments.[14] Circular fields are used *in lieu* of the traditional methods of generating repulsive forces for the PF algorithm and are generated around an obstacle by using its centre of mass. Robots move around the obstacle in the direction of the circular field allowing local minima regions to be avoided. Determining the centre of mass of an obstacle requires its dimensions to be known *a priori*. A PF algorithm is used in conjunction with a fuzzy logic controller (FLC) to navigate a robot through a 2D environment with dynamic obstacles and targets.[17] Two fuzzy logic subsystems are used to reduce the force models and subsequently simplify the required computing power.

A VFF algorithm is introduced which uses a neural network to compensate for uncertainties between robot dynamics for a wheeled robot and the environment.[44] This is extended to create a neural network-based fuzzy algorithm to perform self-tuning for variables, such as the force field size, and to further reduce any uncertainties. A multilevel software architecture uses several different planning algorithms to control an autonomous mower in dynamic, real-time environments.[29] At the highest level, a spiral planner is used to develop a basic global path. The local motion planner employs an RRT algorithm, which takes into consideration vehicle kinematics, to modify the generated path when obstacles are present or a waypoint is infeasible. A VFF algorithm is implemented at a lower

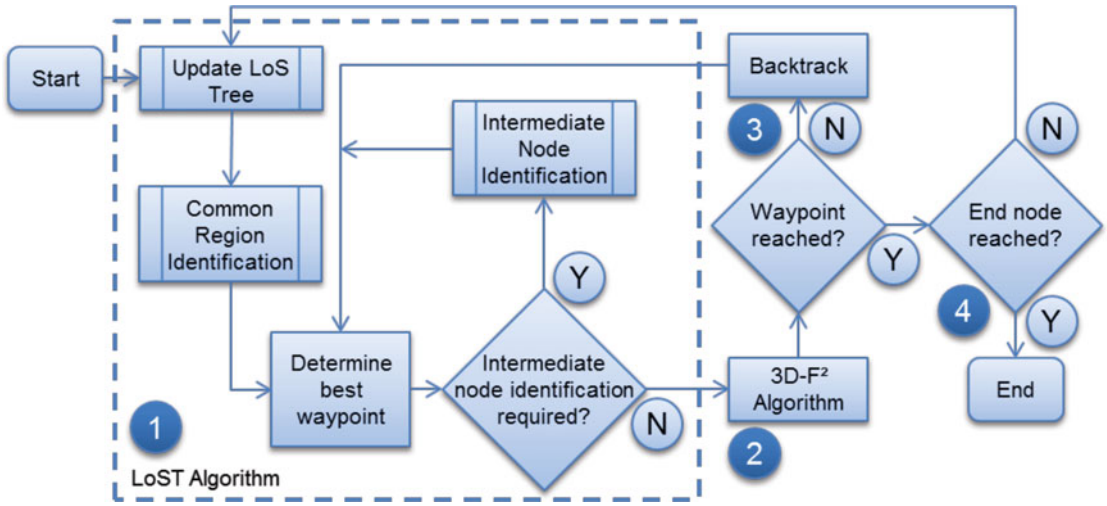


Fig. 3. Basic flow of the approach.

level to avoid dynamic obstacles and correct any errors in the estimated robot pose. A VFF algorithm is combined with a FLC and boundary-follow (BF) algorithm on a simulated, circular mobile robot.[28] In normal operation, the VFF algorithm supplies information, such as obstacle positions and direction to the target, for the FLC to modulate the velocity of the robot. If the robot enters a local minima region, the BF algorithm is invoked and supplies information to the FLC instead. The BF algorithm decides to follow either the left or right wall at a specific distance until the robot escapes the local minima region.

3. An Approach for Planning in Complex Environments

The approach presented consists of two algorithms to find a path from a start node to an end node for an inchworm robot moving in a complex environment. A LoST algorithm searches for the end node and generates waypoints. A 3D-F^2 algorithm finds smooth, collision-free paths for the inchworm robot to follow to each of the generated waypoints. While the 3D-F^2 algorithm has been used, any point-to-point path planning algorithm will work.

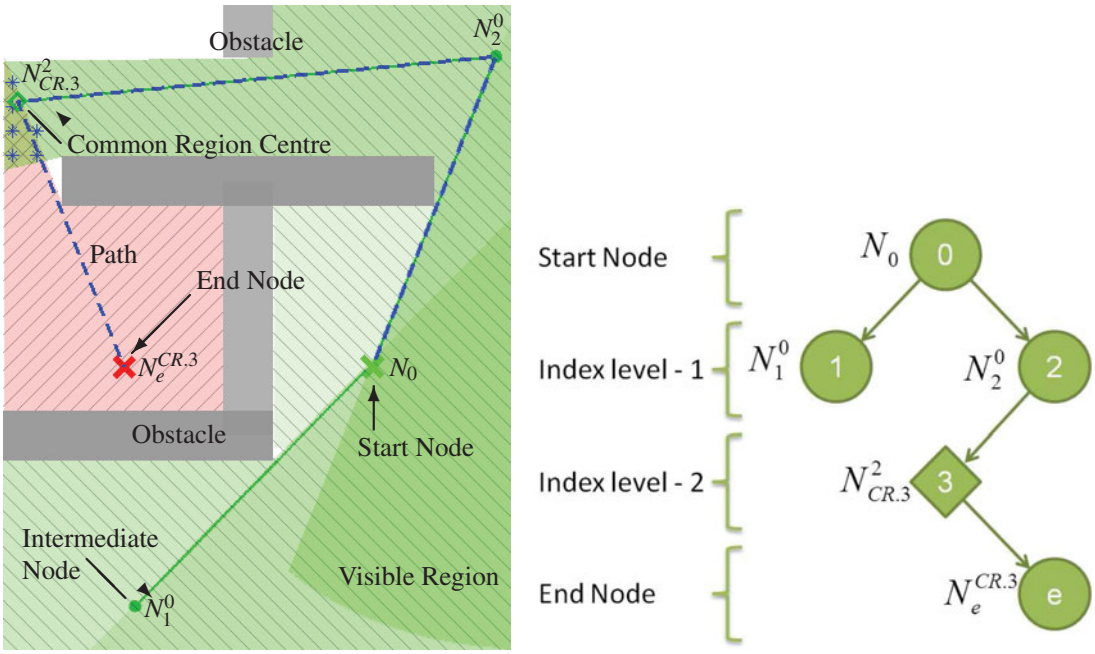
Figure 3 outlines the basic steps in the presented approach. In ①, the LoST algorithm uses sensor readings to search the area surrounding the initial pose and to determine the next best waypoint to move to in an attempt to reach the end node (Section 3.1). If the LoST algorithm fails to find a waypoint, the approach fails. The 3D-F^2 algorithm in ② develops a smooth, collision-free path to the chosen waypoint (Section 3.2). The 3D-F^2 algorithm can fail to find a path if the inchworm robot becomes trapped in a local minima preventing further motion. If this occurs, the inchworm robot backtracks to the pose it started at prior to moving ③. The waypoint is set as inaccessible and the LoST algorithm is reinvoked to find the next best waypoint (Section 3.3). However, if the 3D-F^2 algorithm is successful and the robot has not yet reached the end node, ④, the LoST algorithm is reinvoked. This process is repeated until either the end node is found and a path generated to it or the approach fails (Section 3.1.5).

3.1. Line of sight trees algorithm

The LoST algorithm develops a path in Cartesian space to an end node based on LoS from nodes. The LoS surrounding a node is established by omnidirectionally ray casting around it. From this, both common regions and intermediate nodes are determined. Common regions are areas between which a node and the end node share a common visible region. The centre of this region, the common region centre, is used to connect to the end node. If no common regions are found, an intermediate node is selected to continue the search for the end node. Intermediate nodes are positioned beyond obstacles based on the LoS from a node. Intermediate nodes open new regions to be searched, increasing the chances of finding a common region.

A LoS tree is used to represent the connectivity between the start node, common region centres, intermediate nodes and the end node. By using LoS, an obstacle free path is guaranteed between

https://doi.org/10.1017/S0263574716000047 Published online by Cambridge University Press



(a) LoST algorithm in a simple 2D environment

(b) Related LoS tree for the 2D environment

Fig. 4. (a) The start and end nodes are represented as green and red crosses. From the start node, intermediate nodes, shown as green dots, are searched for until a common region is found, shown as blue asterisks. The centre is determined, the green diamond, and is used to connect to the end node. The final path is shown as a blue dashed line. The green backgrounds with diagonal lines running from the top-left to bottom-right are the start and intermediate node's visible regions. The end node's visible region is represented by the red background with diagonal lines running in the opposite direction. (b) The LoS tree for the shown simple environment. Circles represent nodes which are indexed according to their sorted weighted value at each index level. Diamonds represent common regions and are numerated similarly.

sequential elements of the LoS tree with paths through the environment found by following branches of the LoS tree. The final path is determined by backtracking through the LoS tree from the end node to the common region centre then to each preceding node back to the start node.

The following notation is used to describe the LoST algorithm. The current node, N_j, describes a possible point in Cartesian space, $\{x, y\}$ in 2D and $\{x, y, z\}$ in 3D, with the subscript, j, referring to its index within the LoS tree. An intermediate node, N_i^j, describes a possible point in Cartesian space with the subscript, i, referring to its index while the superscript, j, refers to the index of the node it was generated from. A common region centre, $N_{CR.i}^j$, is a point in Cartesian space with the subscript suffix, i, denoting its index and superscript, j, referring to the index of the node it was found from.

Common regions and intermediate nodes generated from the start node have an index level of 1. The index level of a common region or intermediate node is incremented by 1 from the index level of the node they were generated from. For example, a node with an index level of 3 will generate common region centres with an index level of 4. To prevent the LoST algorithm from searching indefinitely, a maximum index level is used. If a node would generate common region centres at the maximum index level, intermediate nodes cannot also be generated from that node.

Figure 4a shows the LoST algorithm (Algorithm 1) finding a path through an environment with the resulting LoS tree shown in Fig. 4b. For this example, a maximum index level of 2 is used. From both the start and end nodes, N_0 and N_e, the visible regions are determined and checked to find any common regions (Section 3.1.1). The start node's visible region is represented as the green shaded area, while the end node's is red. As no common regions are found between N_0 and N_e, intermediate node identification is performed from the current node, N_0, (Section 3.1.2) resulting in two intermediate nodes, N_1^0 and N_2^0, both of which have an index level of 1. Both intermediate nodes are then weighted (Section 3.1.4) to determine their index within the LoS tree. The LoST algorithm then determines which intermediate node is selected to continue the search process by using a depth

https://doi.org/10.1017/S0263574716000047 Published online by Cambridge University Press

first search on the LoS tree. In this instance, node 1, N_1^0, is selected. The process then continues with common region identification between N_1 and N_e which also fails. Intermediate nodes are not determined as subsequent common region centres will have an index level above the maximum index level. As no further searching is possible from node 1, N_1, its weighting is set to zero and the next best node is determined – which is node 2, N_2. As a common region is found between N_2 and N_e, the common region centre, $N_{CR.3}^2$, is determined and used to connect to the end node. After backtracking through the LoS tree, the final path is from $N_0 \rightarrow N_2^0 \rightarrow N_{CR.3}^2 \rightarrow N_e^{CR.3}$.

Algorithm 1 Line of Sight Trees Algorithm

Input: $StartNode$ ⟵ Start node, $EndNode$ ⟵ End node
Output: $Path$ ⟵ Final path

```
CurrentNode ← StartNode                                   ▷ Set the first node to search from
while Path = ∅ do                                         ▷ Keep searching until a path is found
    CommonRegions ← FindCommonRegions(CurrentNode, EndNode)   ▷ Determine common regions (Alg. 2)
    if CommonRegions = ∅ then                             ▷ If no common regions are found
        IntermediateNodes ← DetermineNextNodes(CurrentNode)   ▷ Determine the intermediate nodes (Alg. 3)
        if IntermediateNodes ≠ ∅ then                     ▷ If intermediate nodes found
            SetNodeWeights(ChildNodes)                    ▷ Weigh the nodes (Section 3.1.4)
            LoSTree ← Connectivity(CurrentNode, IntermediateNodes)   ▷ Update the LoS tree
        end if
        CurrentNode ← DetermineNextNode(LoSTree)          ▷ Use the depth first search to find the next best node
        if CurrentNode = StartNode then                   ▷ If the next index is the start node
            return Path ← ∅                               ▷ The algorithm fails
        end if
    else
        SetNodeWeights(CommonRegions)                     ▷ Weigh the common region centres (Section 3.1.4)
        Path ← backtrack(EndNode, LoSTree, StartNode)     ▷ Backtrack to find the path
    end if
end while
return Path                                               ▷ Successful
```

In the presented approach, the LoST algorithm generates waypoints. These waypoints can either be common region centres or intermediate nodes. Common region centres provide a possible connection to the end node while intermediate nodes provide the next search position for the robot. The waypoint is either the common region centre or intermediate node determined by the depth first search. However, if the robot cannot reach the waypoint, regardless of whether it is a common region centre or an intermediate node, its weighting is set to zero in the LoS tree. The search algorithm used to determine the next best waypoint can be chosen using a search algorithm. A depth first search algorithm ensures a branch of the LoS tree is fully explored; this prevents excessive backtracking. Instead an A* algorithm may be used for the shortest path.

3.1.1. Common region identification algorithm. The visible region of a node, N_j, refers to the visibility of the area surrounding itself within the environment. When the visible region of a node overlaps with that of the end node, N_e, they share a common region (Algorithm 2). The centre of the common region is then used to connect the final path between the node and the end node. The visible region is established by ray casting omnidirectionally (circularly in 2D, spherically in 3D), to a maximum ray cast distance, with the area covered by the rays designating a node's visible region (Fig. 5). While a longer distance will increase a node's visible region, it is prudent to instead set the maximum ray cast distance to the maximum range of a sensor.

To determine common regions, the visible regions from both the current node and the end node are found and discretised into voxels (shown as asterisks in Fig. 5). Discretisation was performed using functionality developed by Paul et al. (2011). An appropriate voxel resolution should be set according to the implementation. However, for maximum visibility it should be ensured that, at the maximum ray cast distance, the distance between ray end points does not exceed the distance between voxels. Any voxels which are visible from both the current node and end node are common voxels. If no common voxels exist, no common region centres are returned. Otherwise common voxels are grouped together to form common regions. This is necessary as an obstacle may split a node's visible region causing multiple common regions. Each common voxel is checked to determine if it is within

https://doi.org/10.1017/S0263574716000047 Published online by Cambridge University Press

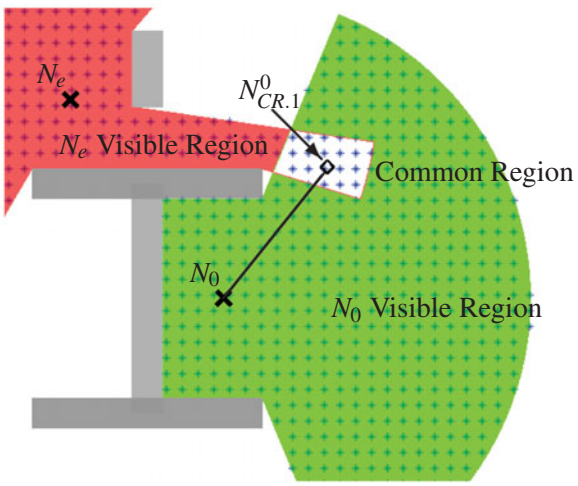


Fig. 5. The start node, N_0, and the end node, N_e, visible regions are shown as green and red regions respectively. The asterisks are the voxels seen from each node. The overlapping area is the common region, shown in white, with the common region centre, $N^0_{CR.i}$, shown as a diamond.

a minimum Euclidean distance of voxels within existing common regions. If it is, all common regions within the distance are merged with the voxel into a single common region. If it is not, a new common region is created from the voxel.

Once all voxels have been evaluated, the common region centres, $N^j_{CR.i}$, of each common region are calculated by finding its geometric centre. All common region centres are then weighted, as described in Section 3.1.4, to determine its index within the LoS tree. This is to differentiate between common region centres when multiple common regions have been found.

In unknown environments, the visible region surrounding the end node cannot be determined. Therefore, the described implementation cannot be directly used. Instead the end node should be searched for directly. To account for this situation in the existing framework, the voxels closest to the end node need to be determined. The common region identification algorithm would then search for these voxels instead of all possible voxels within the end node's visible region. This assumes that the area directly surrounding the end node is free of obstacles and a path can be established from the closest voxels to the end node.

Algorithm 2 Common Region Identification Algorithm

Input: $CurrentNode$ ⟵ Current node, $EndNode$ ⟵ End node
Output: $CommonRegionCentres$ ⟵ Common region centres

```
CurrentVoxels ← Discretise(RayCast(CurrentVisibleRegions))  ▷ Find the visible voxels from the current node
EndVoxels ← Discretise(RayCast(EndVisibleRegions))          ▷ Find the visible voxels from the end node
CommonVoxels ← CompareVoxels(CurrentVoxels, EndVoxels)      ▷ Find any common voxels
if CommonVoxels ≠ ∅ then                                    ▷ If there are common voxels
    for currentVoxel = 1 to CommonVoxels do                 ▷ For each common voxel
        CRInRange ← FindCRInRange(currentVoxel, CRs)        ▷ Find common regions in range of the current voxel
        if CRInRange ≠ ∅ then                               ▷ If there are no common regions within range
            NewCommonRegion(currentVoxel)                   ▷ Create a new common region
        else
            Merge(currentVoxel, CRInRange)                  ▷ Otherwise merge the current voxel with in range common regions
        end if
    end for
    CommonRegionCentres ← FindCentres(CRs)                  ▷ Find the common region centres for each common region
else
    CommonRegionCentres ← ∅                                 ▷ Otherwise there are no common regions
end if
return CommonRegionCentres                                  ▷ Return the common regions
```

https://doi.org/10.1017/S0263574716000047 Published online by Cambridge University Press

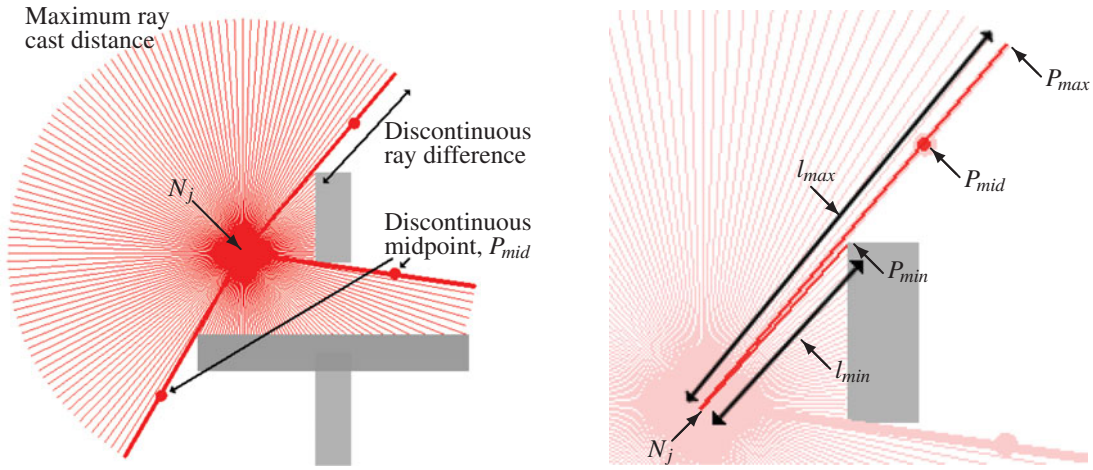


Fig. 6. (a) Ray casting performed at 1° intervals around a central node in 2D with the longer of the discontinuous rays shown as thickened red lines. The centre of the difference between discontinuous rays is taken from the longer to find the discontinuous midpoints. These are shown as red dots along the length of the longer of the discontinuous rays. (b) shows a discontinuous pair with the position of the current node, N_j, the discontinuous midpoint, P_{mid}, the longer ray end point, P_{max}, and length, l_{min} and the shorter ray end point, P_{min}, and length, l_{min}.

3.1.2. Intermediate node identification algorithm. Intermediate nodes are points from where the LoST algorithm searches for common regions. Intermediate nodes are positioned beyond obstacles but within LoS of the current node, N_j. This increases the chances of exploring new regions and successfully finding a common region. Also by maintaining LoS between the current node and its intermediate nodes an obstacle-free path through the environment is ensured.

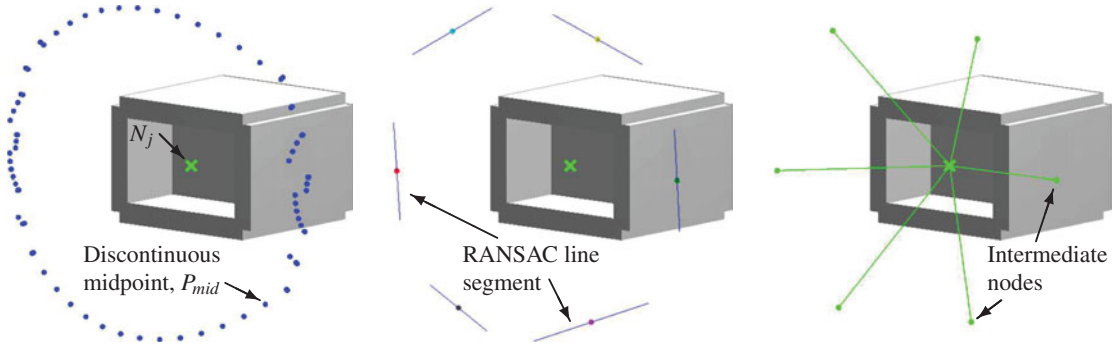
Algorithm 3 is used to determine intermediate nodes for the current node. First, ray casting is performed omnidirectionally (circularly in 2D or spherically in 3D) from the current node to determine the distances to surrounding obstacles. Rays only extend up to a maximum ray cast distance. Figure 6 shows 2D ray casting performed at 1° intervals around a node. The difference between the distance travelled by two adjacent rays is the discontinuous ray difference. If the discontinuous ray difference is longer than a threshold, the two rays are considered to be discontinuous. This implies that an obstruction has occurred between them. A discontinuous midpoint, P_{mid}, is then positioned along the longer ray at a distance from the current node equal to the average length of the two rays, l_{min} and l_{max} (Eq. (1)). Here, P_{max} is the end point of the longer ray. Placing the discontinuous midpoint along the longer ray ensures it extends beyond the obstruction.

Algorithm 3 Intermediate Node Identification Algorithm

Input: $CurrentNode$ ⟵ Current node
Output: $IntermediateNodes$ ⟵ Intermediate nodes

```
RayLengths ← RayCast(CurrentNode)                                  ▷ Determine all ray lengths
DiscontinuousPairs ← FindDiscontinuousPairs(RayLengths)            ▷ Determine the discontinuous pairs
if DiscontinuousPairs ≠ ∅ then                                     ▷ If there are discontinuous pairs
    DiscontinousMidpoints ← FindDiscontinuousMidPoints(DiscontinuousPairs)   ▷ Find the discontinuous midpoints
    if 3DEnvironment then                                          ▷ If it is a 3D environment
        IntermediateNodes ← RANSAC(DiscontinousMidpoints)          ▷ Use RANSAC to find the intermediate nodes
    else
        IntermediateNodes ← DiscontinousMidpoints                  ▷ Use the midpoints as intermediate nodes
    end if
else                                                               ▷ Otherwise if there are no discontinuous pairs
    RayEnds ← FindRayEndPoints(RayLengths)                         ▷ Find the end point of each ray
    DetermineEndWeights(RayEnds)                                   ▷ Weigh the ray ends (Section 3.1.4)
    IntermediateNodes ← Highest(RayEnds)                           ▷ Use the highest weighted ray end as an intermediate node
end if
```

https://doi.org/10.1017/S0263574716000047 Published online by Cambridge University Press



(a) Multiple discontinuous midpoints (b) Line segment midpoints (c) Resultant intermediate nodes

Fig. 7. (a) In 3D environments, LoST algorithm will find a high number of discontinuous midpoints. (b) RANASC is used to find line segment and, subsequently, their midpoints. (c) These midpoints are used as intermediate nodes.

$$P_{mid} = N_j + \frac{l_{min} + l_{max}}{2}(P_{max} - N_j). \tag{1}$$

In 2D environments, the discontinuous midpoints are directly used as the intermediate nodes. However, in 3D environments there will be a large number of discontinuous midpoints along obstacle edges (Fig. 7a). If used directly as intermediate nodes, any common region checks will be similar causing unnecessary computation. Instead resultant midpoints should be found which would encompass the visible regions from the majority of the discontinuous midpoints. As no considerations are made of the environment beyond the current node's visible region, it is assumed that any cluster of discontinuous midpoints sharing a common obstacle edge will have similar visible regions; with the visible region of a discontinuous midpoint at the centre of that cluster having the highest shared visibility with other discontinuous midpoint visible regions within that cluster. Resultant nodes should thus be positioned in the centre of clusters of discontinuous midpoints which share a common obstacle edge. To determine which midpoints share a common edge, a Random Sample Consensus (RANSAC) algorithm is applied to all discontinuous midpoints to determine line segments (Fig. 7b). The geometric centre of these line segments are determined and used as intermediate nodes for the 3D environment (Fig. 7c).

It is possible that in some situations the maximum ray cast distance is insufficient to detect any obstacles resulting in no new intermediate nodes being found. This can prematurely end either a single branch of the LoS tree or the LoST algorithm altogether (Fig. 8a). This can be accounted for by determining the end point for each ray. These points are then weighted (Section 3.1.4) with the highest weighted point used as an intermediate node (Fig. 8b).

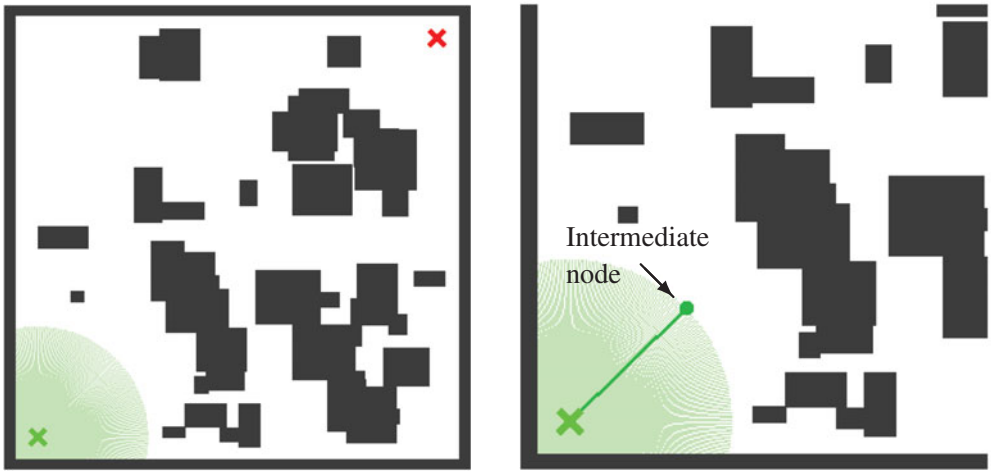
3.1.3. Safe robot workspace. To prevent collisions, the workspace of a robot can be considered in path planning by increasing the size of the obstacles. Instead, by using the ray cast data, an adjusted longer discontinuous ray can be found which will help to avoid collisions. Here, c is the radius of the workspace surrounding the robot.

Determining a new longer discontinuous ray is done in two stages. The first stage adjusts the ray to avoid collisions with the obstacle which caused the discontinuous pair (Fig. 9a). Using the equation of the angle of a chord (Eq. (2)), the minimum angle required to bypass the obstacle, θ_{min}, is obtained. Here, l_{min} is the length of the shorter discontinuous pair. The number of rays in the minimum angle, Z_{off}, is then determined by dividing the minimum angle, θ_{min}, and the angle between rays, θ_{res}, and rounding up the result (Eq. (3)). An offset ray is found by counting from the discontinuous ray the number of rays in the minimum angle (Fig. 9b).

$$\theta_{min} = 2 \arcsin\left(\frac{c}{2l_{min}}\right) \tag{2}$$

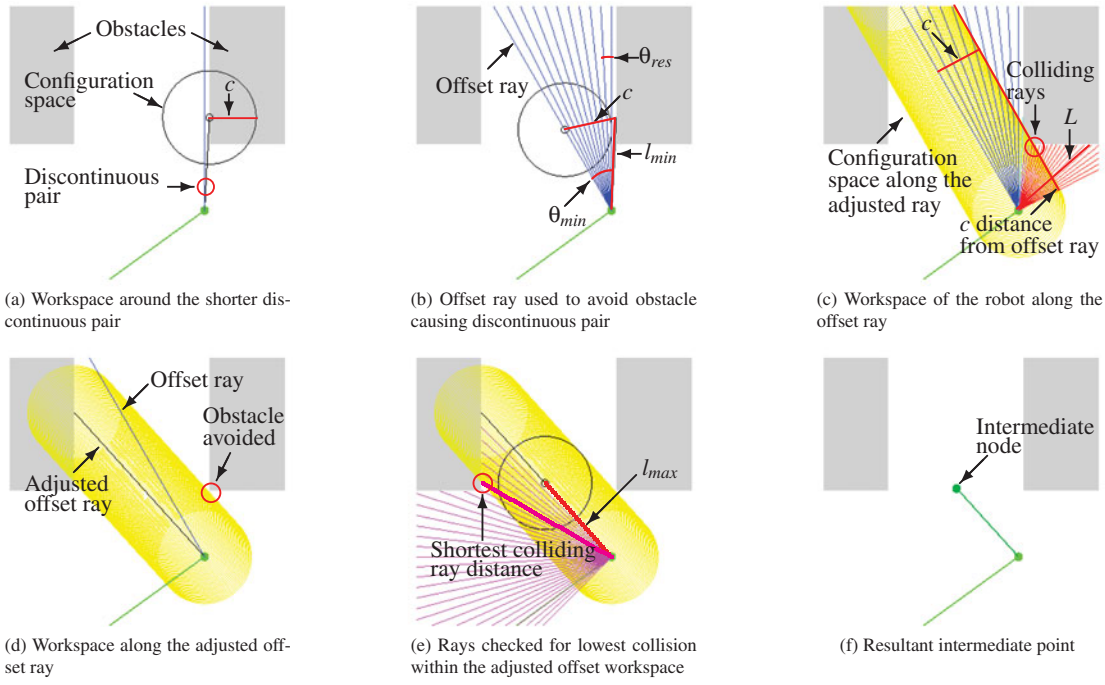
$$Z_{off} = \left\lceil \frac{\theta_{min}}{\theta_{res}} \right\rceil. \tag{3}$$

https://doi.org/10.1017/S0263574716000047 Published online by Cambridge University Press



(a) No obstacles within range (b) Resultant intermediate node

Fig. 8. (a) The maximum ray cast distance may be too short, preventing rays from reaching obstacles. (b) In this case, the highest weighted ray end point should be used as an intermediate node instead.



(a) Workspace around the shorter discontinuous pair

(b) Offset ray used to avoid obstacle causing discontinuous pair

(c) Workspace of the robot along the offset ray

(d) Workspace along the adjusted offset ray

(e) Rays checked for lowest collision within the adjusted offset workspace

(f) Resultant intermediate point

Fig. 9. (a) Workspace around the shorter discontinuous pair should be considered. (b) From the shorter discontinuous pair, the offset representing the number of rays required to avoid the obstacle is determined. (c) The rays colliding with the workspace along the offset ray are determined. (d) The adjusted offset ray avoids the colliding obstacles. (e) The shortest colliding ray within the workspace along the adjusted offset ray is determined. (f) The final discontinuous midpoint based on the shortest colliding ray, minus the workspace, and the original shortest discontinuous ray.

The second stage further offsets the ray to account for collisions prior to the discontinuous pair (Fig. 9c). A collision occurs if a ray ends within the workspace along the offset ray. Only rays prior to the discontinuous pair and up to 90° from the offset ray need be considered; any rays beyond this are not in the direction of motion and will not collide. By using trigonometry, the closest ray is determined and compared to the radius of the workspace, c (Eq. (4)). Here, $\mathbf{L}$ are the ray distances and $\boldsymbol{\theta}$ are their angle from the offset ray. The offset ray is then further adjusted based on the angle

https://doi.org/10.1017/S0263574716000047 Published online by Cambridge University Press

found in Eq. (4) (Fig. 9d).

$$\max_{\mathbf{L}\sin\boldsymbol{\theta}\leq c} \boldsymbol{\theta}. \tag{4}$$

Once the adjusted offset ray has been finalised, the maximum distance along it without colliding with obstacles must be determined (Fig. 9e). Again using trigonometry, the shortest ray which collides with the workspace along the adjusted offset ray is determined (Eq. (5)). In 2D, the rays up to 90° from the adjusted offset ray are checked. In 3D, this also includes all rays up to 90° from the adjusted offset ray in the directions perpendicular to the discontinuous pair. Here, α is the ray's offset angle in the perpendicular direction. The distance of the shortest ray, minus the radius of the workspace, dictates the maximum distance along the adjusted offset ray a robot can travel without colliding, l_{max}. This distance is compared to the shortest discontinuous ray length, again with the radius of the workspace subtracted (Eq. (6)). If it is longer, an intermediate node is found (Eq. (1)) with the end of the longer discontinuous ray, P_{max}, substituted for the end of the adjusted offset ray and the distance of the longer ray, l_{max}, substituted for the distance found in Eq. (5) (Fig. 9f); otherwise no intermediate node is found.

$$\begin{aligned} l_{max} &= \min_{\mathbf{L}\sin\boldsymbol{\theta}\leq c} \mathbf{L} - c \qquad \text{in 2D} \\ l_{max} &= \min_{\sqrt{(\mathbf{L}\cos\boldsymbol{\alpha}\sin\boldsymbol{\theta})^2+(\mathbf{L}\sin\boldsymbol{\alpha})^2}\leq c} \mathbf{L} - c \text{ in 3D.} \end{aligned} \tag{5}$$

$$l_{max} > l_{min} - c. \tag{6}$$

3.1.4. Weighting criteria. Weighting criteria are applied to all intermediate nodes, N_i^j, of the current node, N_j, to improve the chances of selecting intermediate nodes which will converge upon a solution sooner. It also determines which common region centres, $N_{CR.i}^j$, will be used when multiple common region centres are found. In both cases, higher weightings are preferred. While weighting criteria are necessary, the specific criterion used is dependent on the application and is at the discretion of the user. Weighting factors, α_m, can be used to bias certain criteria over others, prioritising nodes based on any environmental factors or to adjust the LoST algorithm to a specific application. The following criteria are all based on the distance from common region centres or intermediate nodes to other nodes within the LoS tree.

The first criterion is based on the Euclidean distance to the end node, N_e (Eq. (7)). The closer intermediate nodes are to the end node, the more likely subsequent common region checks will be successful as it is less likely there will be an obstruction within a shorter distance. The closer common region centres are to the end node, the shorter the final path. This is the criterion necessary for common region centres in the LoST algorithm. The next criterion is based on the Euclidean distance from the start node, N_0 (Eq. (8)). Longer distances improve the chances of intermediate nodes expanding to new regions by searching further away from the start node. The final criterion is based on the Euclidean distance from the current node, N_j (Eq. (9)). The further away an intermediate node is from the current node, the less likely that any new visible region check from the intermediate node will overlap with any regions previously seen, increasing the likelihood of common regions, and subsequently the end node, being found.

$$d_{1.i} = \begin{cases} \|N_i^j - N_e\| & \text{with common region centres} \\ \|N_{CR.i}^j - N_e\| & \text{with intermediate nodes} \end{cases} \tag{7}$$

$$d_{2.i} = \begin{cases} 0 & \text{with common region centres} \\ \|N_{CR.i}^j - N_0\| & \text{with intermediate nodes} \end{cases} \tag{8}$$

$$d_{3.i} = \begin{cases} 0 & \text{with common region centres} \\ \|N_{CR.i}^j - N_j\| & \text{with intermediate nodes} \end{cases}. \tag{9}$$

https://doi.org/10.1017/S0263574716000047 Published online by Cambridge University Press

Once each set of distances, m, have been found, the values are normalised (Eq. (10)). Here, $\min(d_m)$ and $\max(d_m)$ are the minimum and maximum values respectively of each set of distances, d_m.

$$K_{m.i} = \frac{d_{m.i} - \min(d_m)}{\max(d_m) - \min(d_m)}. \tag{10}$$

The final weighting is the sum of individual normalised criteria for each common region (Eq. (11)), and intermediate node (Eq. (12)) multiplied by weighting factors, α. With regards to the end node criterion (Eq. (7)), shorter distances are more favourable. The sum of the weighting factors is 1 (Eq. (13)) to ensure that the final weightings are between 0 and 1.

$$N^j_{CR.i}.w = \alpha_1(1 - K_{1.i}) + \alpha_2 K_{2.i} + \alpha_3 K_{3.i}, \tag{11}$$

$$N^j_i.w = \alpha_1(1 - K_{1.i}) + \alpha_2 K_{2.i} + \alpha_3 K_{3.i}, \tag{12}$$

$$\sum_{m=1}^{3} \alpha_m = 1. \tag{13}$$

3.1.5. Line of sight tree algorithm termination criteria. There are two criteria for terminating the LoST algorithm; a common region centre is found or the depth first search identifies the start node as the next node to search from. If a common region centre is found, the LoST algorithm is successful. The final path is determined by backtracking from the end node to the highest weighted common region centre then, via each subsequent intermediate node, back to the start node.

The second criterion occurs when all nodes of the LoS tree have a zero weighting resulting in the start node being selected as the next node to search from. When this occurs, the LoST algorithm has failed. An intermediate node receives a zero weighting when further searches will cause the current index level to exceed the maximum index level. The maximum index level is used to prevent a branch from being searched indefinitely. The current node's weighting will also be set to zero if all its intermediate nodes have zero weightings. When the LoST algorithm fails either there is no solution or a higher maximum index level must be set.

3.1.6. Updating the line of sight tree algorithm in real-time. The LoST algorithm ray casts omnidirectionally from the current node to determine both intermediate nodes and common regions. In real environments, depth sensors can be used to determine the required information, although omnidirectional ray casting may not be practically applicable due to the limited field of view. Multiple scans may be required to fully capture the necessary distance information surrounding a node.

As the 3D-F^2 algorithm moves the inchworm robot's end-effector to a waypoint generated by the LoST algorithm, further ray cast operations should occur from this waypoint. Optimally, the sensor origin and robot end-effector should be aligned and, while scanning the surrounding environment, the sensor origin position should remain stationary with only its orientation altered. However, sensors are generally offset from the end-effector and often re-orientating will also modify the position. These issues prevent the sensor from ray casting directly from the waypoint. It is possible to find inverse kinematic solutions for each required orientation; however, this can be computationally expensive. Instead a set of poses may be pre-generated which will allow the sensor to completely scan the surrounding environment while minimising the variation in the sensor position. While there may be some overlap with scans, this provides a quick, computationally efficient solution in practical applications.

As the robot moves through each pose, the sensor scans the environment. The readings are applied to the common region identification algorithm (Algorithm 2) to determine a node's visible region and discretised voxels, and to the intermediate node identification algorithm (Algorithm 3) to determine intermediate nodes.

As multiple scans may overlap, the same discretised voxel may be seen multiple times. To account for this, as each scan is taken only new voxels are added to those already found. Similarly, as obstacles may be scanned multiple times due to overlapping scans, multiple intermediate nodes may be found which are within close proximity to each other. This can be minimised by averaging all intermediate nodes within a distance threshold into a new resultant intermediate node.

https://doi.org/10.1017/S0263574716000047 Published online by Cambridge University Press

3.1.7. Additional weighting criteria for an inchworm robot. As the LoST algorithm does not take into account the inchworm robot's kinematics, the robot may not be able to reach the selected waypoint – which can be either a common region centre or an intermediate node. To increase the chances of finding a successful trajectory, the kinematics need to be incorporated into the weighting criteria to bias selections towards waypoints more readily accessible by the inchworm robot, i.e. within its dexterous workspace. To this affect, two additional weighting criteria to those described in Section 3.1.4 are applied.

The first additional criterion is based on the Euclidean distance to the inchworm robot's base, P_b (Eq. (14)). Generally, for a given inchworm robot's workspace, the closer intermediate nodes and common region centres are to the inchworm robot's base the harder they are to reach due to joint hard limits and avoiding self-collisions. However, intermediary nodes and common region centres too far from the base also may not be reachable. The second additional criterion is then based on the Euclidean distance to the boundary of the inchworm robot's workspace. The number of valid poses to reach an intermediate node or common region centre decreases the closer either are to the boundary of the workspace. As such selecting an intermediary node or common region centre further away from the boundary would be more likely to find a valid pose. The workspace boundary can be difficult to calculate due to possible joint orientations and limits. This calculation is simplified by first determining the distance from the intermediate node and common regions to the base, P_b. This is then subtracted from the inchworm robot's length, d_w (Eq. (15)).

$$d_{4.i} = \begin{cases} \|N_i^j - P_b\| & \text{with common region centres} \\ \|N_{CR.i}^j - P_b\| & \text{with intermediate nodes} \end{cases}, \tag{14}$$

$$d_{5.i} = \begin{cases} d_w - \|N_i^j - P_b\| & \text{with common region centres} \\ d_w - \|N_{CR.i}^j - P_b\| & \text{with intermediate nodes} \end{cases}. \tag{15}$$

Each set of distances, m, are then normalised (Eq. (10)). The final weighting is the sum of the normalised criteria for each common region (Eq. (16)), and intermediate node (Eq. (17)), multiplied by weighting factors, α. The weightings factors should be modified so their sum equals 1 (Eq. (18)) ensuring the final weightings are between 0 and 1.

$$N_{CR.i}^j.w = \alpha_1(1 - K_{1.i}) + \alpha_2 K_{2.i} + \alpha_3 K_{3.i} + \alpha_4 K_{4.i} + \alpha_5 K_{5.i}, \tag{16}$$

$$N_i^j.w = \alpha_1(1 - K_{1.i}) + \alpha_2 K_{2.i} + \alpha_3 K_{3.i} + \alpha_4 K_{4.i} + \alpha_5 K_{5.i}, \tag{17}$$

$$\sum_{m=1}^{6} \alpha_m = 1. \tag{18}$$

3.2. 3-dimensional force field algorithm

A 3D-F^2 algorithm[5] is used to find smooth, collision-free paths for the inchworm robot to follow between waypoints supplied by the LoST algorithm. The 3D-F^2 algorithm uses a combination of attractive and repulsive forces to guide the end-effector towards a goal position while simultaneously moving away from obstacles within the environment. The 3D-F^2 algorithm has been included to provide completeness to the approach; however, any point-to-point manipulator path planning algorithm can be used instead. For additional details regarding the 3D-F^2 algorithm, refer to Chotiprayanakul et al (2012).

3.2.1. Kinematics of the inchworm robot. The kinematics of the inchworm robot from a coordinate frame, $n-1$, to the next coordinate frame, n, are described using a homogeneous transformation matrix, ${}^{n-1}T_n$ (Eq. (19)). Each homogeneous transformation matrix is comprised of a rotation matrix, ${}^{n-1}R_n$, and a position vector, ${}^{n-1}P_n$, which are described using Denavit–Hartenberg parameters. By multiplying successive homogeneous transformation matrices from the base coordinate frame to a joint's coordinate frame, j, that joint's position, P_j, and rotation, R_j, can be extracted from the

https://doi.org/10.1017/S0263574716000047 Published online by Cambridge University Press

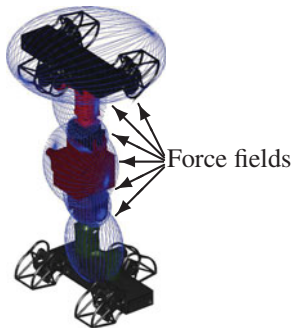


Fig. 10. Blue ellipsoids surroundings links of the simulated 7DOF inchworm robot to represent the force fields.

resulting homogeneous transformation matrix (Eq. (21)).

$$^{n-1}T_n = \begin{bmatrix} ^{n-1}R(q_n)_n & ^{n-1}P_n \\ 0_{1x3} & 1 \end{bmatrix}, \tag{19}$$

$$q = \begin{bmatrix} q_1 & q_2 & q_3 & q_4 & q_5 & q_6 & q_7 \end{bmatrix}^T, \tag{20}$$

$$^0T_j(q) = \prod_{n=1}^{j} {}^{n-1}T_n(q_n). \tag{21}$$

The dynamic model of the inchworm robot is given in Eq. (22). Here, I are the link inertias, β are the joint damping-friction constants, k_{sp} are the joint spring constants and $\ddot{q}$, $\dot{q}$, q are the joint accelerations, velocities and positions respectively. $\mathbf{\Gamma}$ is the virtual torque-force matrix which contains the joint torque components, τ_j, for each joint (Eq. (23)). The virtual torque-force matrix for the inchworm robot is calculated by combining the virtual torque-force matrices generated from the attractive force, F_{att}, and the repulsive forces, F_{rep} (Eq. (24)). Here, H_n are matrice which transforms the force from Cartesian space to joint space.

$$\mathbf{\Gamma} = I\ddot{q} + \beta\dot{q} + k_{sp}q, \tag{22}$$

$$\mathbf{\Gamma} = \begin{bmatrix} \tau_1 & \tau_2 & \tau_3 & \tau_4 & \tau_5 & \tau_6 & \tau_7 \end{bmatrix}^T, \tag{23}$$

$$\mathbf{\Gamma} = H_7 F_{att} + \sum_{n=1}^{7} H_n F_{rep_n}. \tag{24}$$

3.2.2. Attractive force. The attractive force is used to guide the inchworm robot's end-effector, P_e, and the goal position, P_t, generated by the LoST algorithm as a waypoint. A waypoint is either an intermediary node or a common region centre. This is calculated by determining the distance between the two points and applying a sigmoid function to the resultant (Eq. (25)). Here, K_{att} is the maximum attractive force, K_{zero} is a small non-zero positive constant and K_s is a constant which determines how the attractive force varies over distance.

$$F_{att} = K_{att} / \left(1 + \frac{\exp\left(-K_s|P_t - P_e|\right)}{K_{zero}}\right) \frac{P_t - P_e}{|P_t - P_e|}. \tag{25}$$

3.2.3. Repulsive force. Repulsive forces are used to prevent the inchworm robot from colliding with the environment. In this approach, obstacles are represented as a series of environmental voxels. These environmental voxels can be determined using sensor scans.[31] Repulsive forces are generated when an environmental voxel enters within range of virtual ellipsoids which surround each link of the inchworm robot (Fig. 10). The further an environmental voxel enters into an ellipsoid the higher the repulsive force generated.

https://doi.org/10.1017/S0263574716000047 Published online by Cambridge University Press

Repulsive forces are generated for each robot link, F_{rep_n}. The force is calculated by first determining the distance the closest environmental voxel, P_{obs}, enters into an ellipsoid, C_x. If this distance is less than the ellipsoid coverage constant, K_P, plus a distance factor, K_d, then a repulsive force is calculated based on a sigmoid function (Eq. (26)). This force is applied to the closest point on the ellipsoid's link, P_{cls}, to the environmental voxel, P_{obs}. Here, K_f is the coefficient of amplitude.

$$F_{rep_n} = K_f - K_f / \left(1 + \exp\left(-10 \frac{Cx - K_P - 0.5K_d}{K_d}\right)\right) \frac{P_{cls} - P_{obs}}{|P_{cls} - P_{obs}|}. \tag{26}$$

3.3. Reinvoking the line of sight trees algorithm

The 3D-F^2 algorithm may fail to find a successful path if the inchworm robot enters a local minima region. When this occurs, the weighting of the waypoint attempting to be reached is set to zero and the LoST algorithm is reinvoked to find the next best waypoint. A waypoint can be either an intermediate node or a common region centre. Once found, the branches of the previous and new best waypoints on the LoS tree are compared to determine the node at which they intersect. The inchworm robot backtracks to the pose it was at when previously at the intersection node. This prevents the inchworm robot from moving along non-LoS paths. The 3D-F^2 algorithm then continues as normal.

Local minima occur when the sum of forces acting on the inchworm robot equal zero, essentially stalling motion. However, this is not necessarily instantaneous. While the forces acting may equate to zero, the inchworm robot's momentum may cause it to continue to move through and potentially exit the local minima region. Also, the inchworm robot may be caught in an area which results in continuous, repeated motion such as bouncing between two obstacles. The inchworm robot is then considered to be in a local minima region when it is not at the end node and at least one of the following three conditions are met.

The first condition occurs when the inchworm robot has not moved significantly over a sample period. This occurs when the sum of the variation between consecutive joint positions, q, along the path over a sample period, k_{sample}, is less than a threshold, k_q (Eq. (27)). Taking the values over a sample period prevents erroneously detecting local minima when the inchworm robot changes direction as it may momentarily stops moving.

$$\sum_{l=2}^{k_{sample}} |q_{l-1} - q_l| < k_q. \tag{27}$$

The second local minima condition occurs when the joint velocities over a set period are insignificant (Eq. (28)). This detects local minima regions sooner than the first condition when joint accelerations are approaching zero. This occurs when the sum of the variation between consecutive joint velocities, $\dot{q}$, over the sample period, k_{sample}, is less than the velocity threshold, $k_{\dot{q}}$.

$$\sum_{l=2}^{k_{sample}} |\dot{q}_{l-1} - \dot{q}_l| < k_{\dot{q}}. \tag{28}$$

The final condition is used to identify when the inchworm robot is continuously repeating the same motions (Eq. (29)). This is defined as when both the current pose, q, and velocity, $\dot{q}$, have occurred at the same instance previously along the pose set, q_{set}, and velocity set, $\dot{q}_{set}$, within a set of tolerances, $k_{q_{set}}$ and $k_{\dot{q}_{set}}$.

$$|q \in q_{set}| < k_{q_{set}} \wedge |\dot{q} \in \dot{q}_{set}| < k_{\dot{q}_{set}}. \tag{29}$$

4. Results

Simulations were performed using MatLab® and were designed to test the capabilities of both the LoST algorithm and the presented approach. An RRT algorithm[6] was used as a comparison in these experiments. The approach was tested using a 7DOF inchworm robot.[43]

https://doi.org/10.1017/S0263574716000047 Published online by Cambridge University Press

Table I. Unknown environment with 40 random obstacles – averaged over 100 runs.

Type	$LoST_5$	RRT
Completion rate	100%	100%
Average nodes in final path	3.98	7.87
Average nodes found in total	304.96	574.43

The RRT algorithm utilises multiple trees to improve convergence towards a solution. This method has increased success in finding paths through narrow passageways, and with fewer collision checks and in fewer search iterations than standard RRT algorithms.[6] The RRT algorithm has been allowed to propagate from trees at both the start and end nodes to increase the chances of converging upon a successful solution. To prevent the RRT algorithm from searching indefinitely, the maximum number of search iterations was set to 2000.

When the LoST algorithm is specified with a subscript, this subscript refers to the maximum node level; for example, $LoST_3$ refers to a LoST algorithm run with a maximum index level of 3. In all experiments, the maximum ray cast distance was set to ensure that the end node could not be reached from the start node. The final path length is the distance a simulated point robot travels along the found path, while the total distance travelled is the distance a simulated robot moves between each node while searching for the end node using the LoST algorithm. The final path length gives an indication of the distance required to reach the end node if the environment was known and searched prior to moving, while the total distance travelled indicates the distance travelled if the environment is unknown. All values are averaged over the number of times the algorithms are run. When an algorithm fails to find a solution, its data set is not included in the average.

4.1. Randomly generated unknown 2D environment

The LoST algorithm was first tested in 100 unknown environments with 40 randomly sized rectangular obstacles positioned randomly throughout (Fig. 11). The start and end nodes are known. The unknown space is represented in the figure by a grey overlayed grid. As the LoST algorithm searches, the visible regions from each node reveals the environment beneath the overlayed grid (Figs. 11a–11e). The LoST algorithm's maximum index level is set to 5. For comparison, the RRT algorithm is used which was run three times on each generated environment to prevent biased results due to its probabilistic nature. The RRT algorithm is also assumed to know the environment.

Table I shows the results for both the LoST and RRT algorithms. Both algorithms had a 100% completion rate. Results show that the $LoST_5$ algorithm found a path with an average of 50.5% less nodes in the final path, despite functioning in an unknown environment. This is intrinsic to the LoST algorithm as ray casting does not rely on any previous knowledge of the environment. The only differentiation between known and unknown environments is the termination criteria. In known environments, the LoST algorithm searches for common regions. Alternatively in unknown environments, the current node searches for a voxel directly surrounding the end node, which is subsequently used to connect the end node to the LoS tree. The final path is still revealed by backtracking through the LoS tree. In unknown environments, this effectively increases the maximum number of nodes required to successfully find a path by one in comparison to known environments.

4.2. Randomly generated known 2D and 3D environments

The LoST algorithm is then tested in known environments. The environments are generated by creating a number of randomly sized rectangular obstacles in 2D, or rectangular prisms in 3D, positioned randomly within the area. Example environments are shown in Figs. 12a and 13a. Obstacles were not generated near the robot's predetermined start and end locations.

For both 2D and 3D environments, the LoST and RRT algorithms were run on six sets of 100 randomly generated environments. Beginning with 20 obstacles, each subsequent set incrementally generated an additional 20 obstacles with the last set containing 120 obstacles. For each set of environments, the LoST algorithm was run from a maximum index level of 1 ($LoST_1$) to a maximum index level of 5 ($LoST_5$). Additionally, in 3D environments the LoST algorithm and the RRT algorithms were called three times in each environment to account for their probabilistic natures

https://doi.org/10.1017/S0263574716000047 Published online by Cambridge University Press

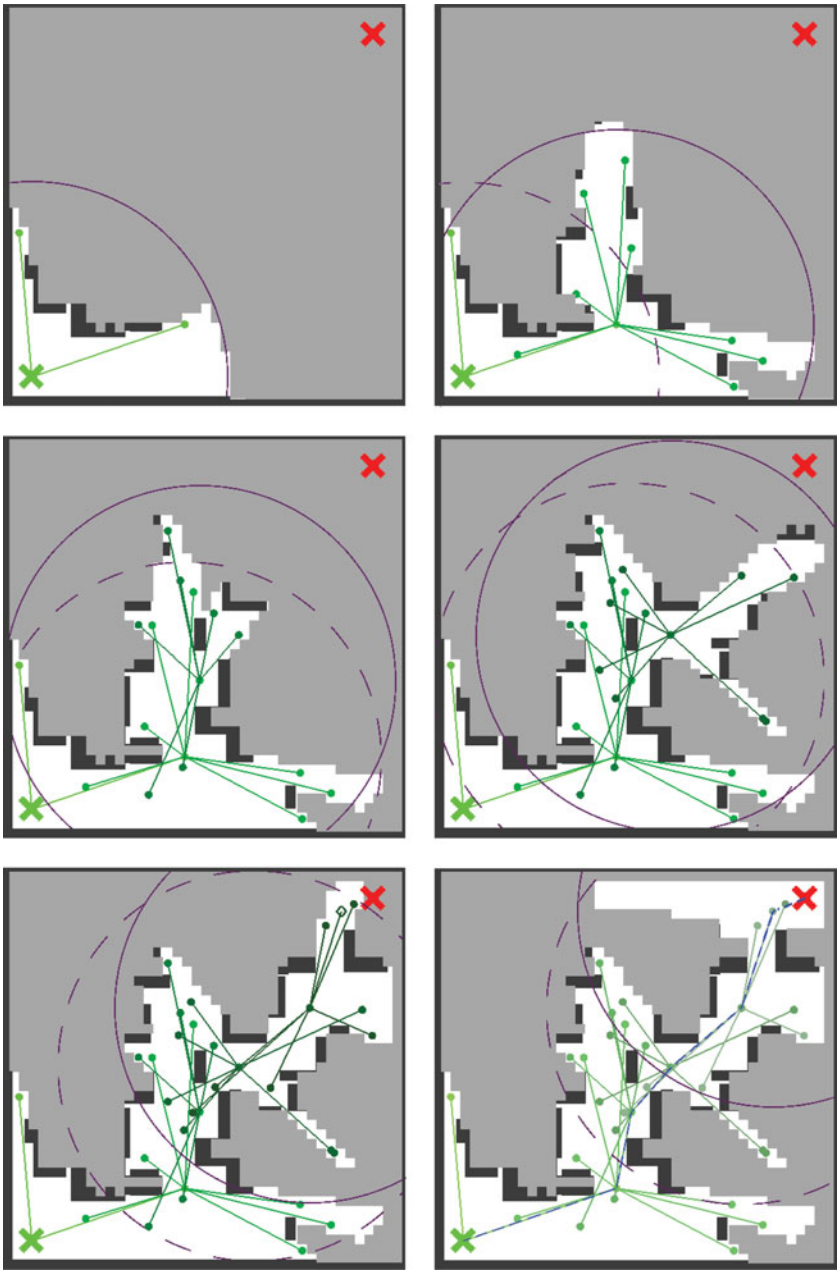
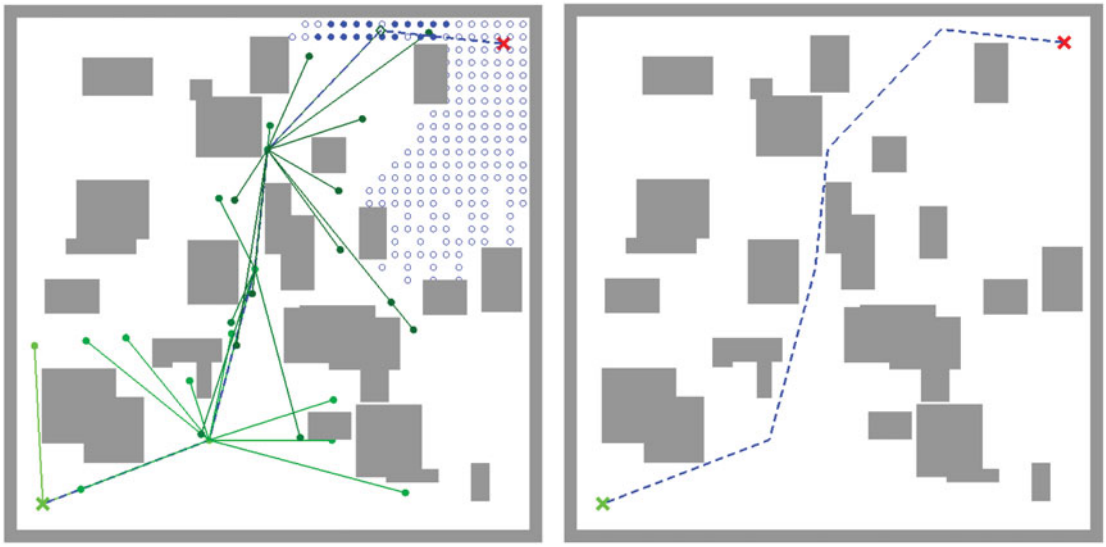


Fig. 11. An example randomly generated unknown 2D environment with 40 random obstacles. Unknown space, which is represented as grey regions, is uncovered based on the visible region of a node. The green and red crosses are the start and end node respectively with the LoS tree and intermediate nodes shown as green lines and dots. The final path is shown as a blue dashed line. At each index, the maximum ray cast distance is shown as a solid magenta circle, with a broken circle denoting the previous maximum ray cast distances. (a) First Node Index. (b) Second Node Index. (c) Third Node Index. (d) Fourth Node Index. (e) Fifth Node Index. (f) Final Path.

with all successful results recorded. The results for the 2D environments are shown in Tables II–VII with the results for the 3D environments shown in Tables VIII–XIII.

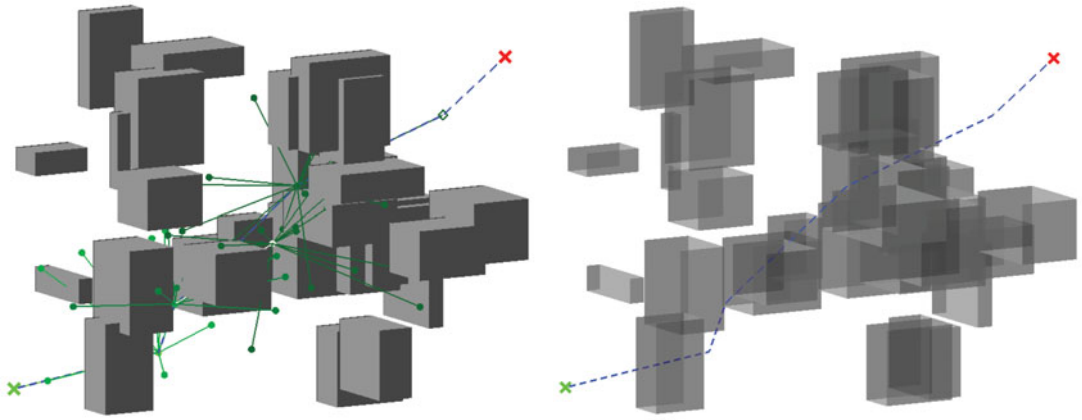
The maximum index level highly influences the tested metrics associated with the LoST algorithm. The maximum index level specifies how far along a branch of a LoS tree is searched before terminating. At higher levels, branches are searched further, increasing the chances of finding a common region and a solution. Although while higher maximum index levels do yield higher completion rates, the LoST algorithm may exhaustively search regions within the environment, such as with obstacles in close proximity forming barriers, which will not yield a successful solution. In this case, a lower

https://doi.org/10.1017/S0263574716000047 Published online by Cambridge University Press



(a) Complete LoS tree (b) Final full path

Fig. 12. An example 2D environment with (a) the LoS tree and common regions and (b) the final LoST path. The start node is represented as a green X, end node a red X, green dots and their connecting lines are intermediate nodes, the empty blue circles represent the visible region from the end node, the filled blue circles are common regions and the blue dotted line is the final path found through the environment.



(a) Complete LoS tree (b) Final full path

Fig. 13. (a) An example 3D environment with the LoS tree and (b) the final path found. The start node is represented as a green X, end node a red X, green dots and their connecting lines are intermediate nodes and the blue dotted line is the final path found through the environment.

maximum index level would be beneficial so as to prevent a large number of collision checks and excessive total distance travelled. However, if the maximum index level is too low the search may end prematurely before a path is found. An appropriate maximum index level should then be selected which gives a high success rate and maintains a reduced number of total nodes found and total distance travelled. In the 3D environments (Tables VIII to XIII), it can be seen that at certain maximum index levels with high completion rates, some LoST algorithm's find solutions with a reduced total number of nodes found and total distance travelled compared to others; with 20 and 40 obstacles this occurs at maximum index level 3; with 60, 80 and 100 obstacles at a maximum index level 4; and with 120 obstacles at maximum index level 5. In 2D environments (Tables II–VII), this is less evident as completion rates are not considerably high.

In the majority of cases, the LoST algorithm develops paths with fewer nodes than the RRT algorithm with the variation between the two algorithms more obvious with an increasing number of obstacles. In 2D environments, at a minimum the RRT algorithm requires on average more than twice

https://doi.org/10.1017/S0263574716000047 Published online by Cambridge University Press

Table II. 2D environments with 20 random obstacles – averaged over 100 runs. LoST algorithm run at various maximum index levels.

Type	$LoST_1$	$LoST_2$	$LoST_3$	$LoST_4$	$LoST_5$	RRT
Completion rate	44.00%	94.00%	97.00%	97.00%	97.00%	93.92%
Average nodes in final path	1.00	1.70	1.99	2.10	2.20	4.80
Average total number of nodes found	28.27	29.50	35.11	34.61	33.52	102.99
Average collision checks	1423.64	1508.94	1825.98	1803.71	1762.89	1316.97
Average final path length	2.70 m	2.93 m	3.04 m	3.08 m	3.12 m	3.43 m
Average total distance travelled	4.35 m	4.06 m	5.11 m	4.56 m	4.37 m	–

Table III. 2D environments with 40 random obstacles – averaged over 100 runs. LoST algorithm run at various maximum index levels.

Type	$LoST_1$	$LoST_2$	$LoST_3$	$LoST_4$	$LoST_5$	RRT
Completion rate	5.00%	50.00%	84.00%	91.00%	93.00%	81.17%
Average nodes in final path	1.00	1.98	2.65	3.26	3.31	8.49
Average total number of nodes found	34.20	82.90	118.39	317.53	810.70	465.26
Average collision checks	1512.00	3988.80	5922.86	15436.48	39374.19	2403.79
Average final path length	2.67 m	3.13 m	3.25 m	3.50 m	3.55 m	3.69m
Average total distance travelled	4.77 m	17.27 m	23.90 m	69.18 m	176.72 m	–

Table IV. 2D environments with 60 random obstacles – averaged over 100 runs. LoST algorithm run at various maximum index levels.

Type	$LoST_1$	$LoST_2$	$LoST_3$	$LoST_4$	$LoST_5$	RRT
Completion rate	0.00%	13.00%	45.00%	67.00%	75.00%	51.33%
Average nodes in final path	–	2.00	2.78	3.76	4.36	10.77
Average total number of nodes found	–	71.62	362.69	1262.01	3263.13	952.99
Average collision checks	–	3793.85	19440.00	68555.82	169419.20	3866.96
Average final path length	–	3.18 m	3.37 m	3.64 m	3.82 m	3.81m
Average total distance travelled	–	15.46 m	85.58 m	298.29 m	731.58 m	–

Table V. 2D environments with 80 random obstacles – averaged over 100 runs. LoST algorithm run at various maximum index levels.

Type	$LoST_1$	$LoST_2$	$LoST_3$	$LoST_4$	$LoST_5$	RRT
Completion rate	0.00%	2.00%	5.00%	36.00%	41.00%	15.92%
Average nodes in final path	–	2.00	3.00	3.86	4.71	10.56
Average total number of nodes found	–	36.00	212.60	1035.72	2432.98	1091.35
Average collision checks	–	2340.00	14040.00	67030.00	150989.27	4282.06
Average final path length	–	3.17 m	3.25 m	3.58 m	3.88 m	3.79m
Average total distance travelled	–	7.89 m	57.09 m	287.40 m	605.93 m	–

the number of nodes in the final path compared to the LoST algorithm, while in 3D environments, the variation between the two algorithms is lower as the relative amount of free space is increased. The RRT algorithm does find paths with fewer nodes in the 20 obstacle case; in all other cases the LoST algorithm's paths have fewer. As the concentration of obstacles increases, both algorithms require more nodes to find the final path. By reducing the amount of free space, finding interconnecting nodes becomes more difficult for the RRT algorithm which increases the number of nodes required to find a path. The LoST algorithm has a decreased chance of finding common regions as a node's visible region is more hindered by additional obstacles which similarly increases the number of nodes required. However, significantly more nodes are found as more obstacle edges are now visible. While this may be necessary to navigate through complex environments, it does increases computational

https://doi.org/10.1017/S0263574716000047 Published online by Cambridge University Press

Table VI. 2D environments with 100 random obstacles – averaged over 100 runs. LoST algorithm run at various maximum index levels.

Type	$LoST_1$	$LoST_2$	$LoST_3$	$LoST_4$	$LoST_5$	RRT
Completion rate	0.00%	0.00%	1.00%	8.00%	25.00%	3.75%
Average nodes in final path	–	–	3.00	3.75	4.68	9.84
Average total number of nodes found	–	–	115.00	513.13	4607.60	1098.82
Average collision checks	–	–	6480.00	35370.00	280411.20	4304.45
Average final path length	–	–	3.49 m	3.51 m	3.72 m	3.78m
Average total distance travelled	–	–	26.46 m	156.40 m	1153.36 m	–

Table VII. 2D environments with 120 random obstacles – averaged over 100 runs. LoST algorithm run at various maximum index levels.

Type	$LoST_1$	$LoST_2$	$LoST_3$	$LoST_4$	$LoST_5$	RRT
Completion rate	0.00%	0.00%	1.00%	3.00%	10.00%	1.58%
Average nodes in final path	–	–	3.00	4.00	4.90	9.53
Average total number of nodes found	–	–	98.00	1103.33	1620.00	1146.90
Average collision checks	–	–	6480.00	76800.00	108756.00	4448.71
Average final path length	–	–	3.36 m	3.40 m	3.61 m	3.75m
Average total distance travelled	–	–	23.41 m	320.51 m	419.78 m	–

Table VIII. 3D environments with 20 random obstacles – averaged over 100 runs. LoST algorithm run at various maximum index levels.

Type	$LoST_1$	$LoST_2$	$LoST_3$	$LoST_4$	$LoST_5$	RRT
Completion rate	0.00%	100.00%	100.00%	100.00%	100.00%	100.00%
Average nodes in final path	–	2.00	2.18	2.22	2.21	2.13
Average total number of nodes found	–	80.53	61.63	60.54	61.32	3.11
Average collision checks	–	7352.64	6153.80	6376.32	6912.00	1017.34
Average final path length	–	2.79 m	2.83 m	2.88 m	2.87 m	3.14 m
Average total distance travelled	–	7.06 m	4.59 m	4.49 m	4.83 m	–

Table IX. 3D environments with 40 random obstacles – averaged over 100 runs. LoST algorithm run at various maximum index levels.

Type	$LoST_1$	$LoST_2$	$LoST_3$	$LoST_4$	$LoST_5$	RRT
Completion rate	0.00%	100.00%	100.00%	100.00%	100.00%	100.00%
Average nodes in final path	–	2.00	2.22	2.24	2.23	3.15
Average total number of nodes found	–	165.53	81.95	85.01	90.67	7.05
Average collision checks	–	10874.94	6596.64	7758.72	10640.16	1029.15
Average final path length	–	2.77 m	2.84 m	2.86 m	2.83 m	3.38 m
Average total distance travelled	–	12.40 m	4.93 m	5.45 m	6.52 m	–

times. This allows the LoST algorithm to intrinsically direct paths around obstacles allowing it to converge upon a solution with fewer nodes in the final path as opposed to the RRT algorithm.

In comparison to lower maximum index levels, when a LoST algorithm is run with a higher maximum index level, the number of nodes in the final path will be higher. Similar LoS trees will be generated on the same environment with LoST algorithms using higher maximum index levels mimicking paths followed by that of LoST algorithms with lower levels. Any common region found by the lower level LoST algorithm will also be found by the higher and will terminate at the same time. However, the higher will also potentially find common regions by searching further along LoS tree branches. This will increase the number of nodes in the final path.

For both algorithms, the completion rate similarly is related the amount of free space in the environment. As the amount of free space decreases, connecting nodes becomes more difficult for

https://doi.org/10.1017/S0263574716000047 Published online by Cambridge University Press

Table X. 3D environments with 60 random obstacles – averaged over 100 runs. LoST algorithm run at various maximum index levels.

Type	$LoST_1$	$LoST_2$	$LoST_3$	$LoST_4$	$LoST_5$	RRT
Completion rate	0.00%	94.33%	99.00%	100.00%	100.00%	98.50%
Average nodes in final path	–	2.00	2.36	2.33	2.39	3.86
Average total number of nodes found	–	245.58	100.73	94.40	101.99	11.71
Average collision checks	–	14778.06	7780.36	7655.04	11491.20	1043.13
Average final path length	–	2.79 m	2.88 m	2.88 m	2.90 m	3.55 m
Average total distance travelled	–	18.18 m	6.16 m	5.50 m	7.08 m	–

Table XI. 3D environments with 80 random obstacles – averaged over 100 runs. LoST algorithm run at various maximum index levels.

Type	$LoST_1$	$LoST_2$	$LoST_3$	$LoST_4$	$LoST_5$	RRT
Completion rate	0.00%	93.00%	99.00%	99.00%	100.00%	98.00%
Average nodes in final path	–	2.00	2.50	2.63	2.63	4.39
Average total number of nodes found	–	380.09	165.11	111.56	108.16	19.27
Average collision checks	–	22213.89	11144.73	10573.09	12765.60	1065.82
Average final path length	–	2.80 m	2.92 m	2.98 m	2.97 m	3.63 m
Average total distance travelled	–	28.89 m	10.47 m	7.24 m	8.23 m	–

Table XII. 3D environments with 100 random obstacles – averaged over 100 runs. LoST algorithm run at various maximum index levels.

Type	$LoST_1$	$LoST_2$	$LoST_3$	$LoST_4$	$LoST_5$	RRT
Completion rate	0.00%	82.33%	99.00%	99.00%	100.00%	81.00%
Average nodes in final path	–	2.00	2.67	2.85	2.86	5.20
Average total number of nodes found	–	1036.88	274.94	117.13	117.75	37.71
Average collision checks	–	58239.64	19005.88	10763.94	13193.28	1086.34
Average final path length	–	2.78 m	3.00 m	3.05 m	3.02 m	3.83 m
Average total distance travelled	–	81.11 m	18.65 m	7.75 m	8.31 m	–

Table XIII. 3D environments with 120 random obstacles – averaged over 100 runs. LoST algorithm run at various maximum index levels.

Type	$LoST_1$	$LoST_2$	$LoST_3$	$LoST_4$	$LoST_5$	RRT
Completion rate	0.00%	64.33%	95.33%	96.00%	97.00%	74.70%
Average nodes in final path	–	2.00	2.81	3.05	3.11	5.45
Average total number of nodes found	–	1308.98	437.51	213.66	128.10	46.61
Average collision checks	–	77750.59	29162.26	17153.28	16774.56	1113.91
Average final path length	–	2.77 m	3.03 m	3.10 m	3.11 m	3.77 m
Average total distance travelled	–	107.99 m	31.10 m	14.67 m	10.53 m	–

the RRT algorithm and it is more difficult for the LoST algorithm to find common regions. However, as more obstacles are introduced, the LoST algorithm finds a higher number of intermediate nodes which can improve the completion rate. As previously mentioned, the maximum index level highly influences the LoST algorithm. While higher maximum index levels will result in higher completion rates, a minimum level is necessary to find solutions. As seen in the results when the minimum level is too low, the LoST algorithm terminates prematurely significantly reducing the completion rate or potentially finding no solution at all.

The large number of total nodes found by the LoST algorithm also occurs due to the indiscriminate nature of the ray cast operation. As the ray casting is omnidirectional with no considerations made to the location of the current node, subsequent intermediate node checks may find new intermediate nodes in close proximity to the current node or other nodes within the tree. Subsequently, the visible

https://doi.org/10.1017/S0263574716000047 Published online by Cambridge University Press

Table XIV. 3D complex partitioned tunnel – averaged over 100 runs.

Type	LoST	RRT
Completion rate	100%	100%
Average nodes in final path	5.00	27.28
Average nodes found in total	35.20	759.4
Average collision checks	8479.17	2372.09
Average final path length	5.24 m	5.54 m
Average total distance travelled	5.24 m	–

regions from these nodes may be similar when compared to nodes within their proximity, thus providing limited new information. This may be beneficial in some cases where a slight change in position will reveal additional, unseen regions. However, in the majority of cases it will cause unnecessary computation.

The LoST algorithm performs a significantly larger number of collision checks when compared to the RRT algorithm. The RRT algorithm performs collision checks only when attempting to connect nodes to an existing tree, whereas each individual ray cast operation is a single collision check with omnidirectional ray casting required to determine a node's visible region. To minimise the number of collision checks, the resolution between rays should be specified to minimise the total number of ray casting operations while ensuring the surrounding region is completely visible. In 3D environments, the number of collision checks increases as the visible region is volumetric.

The final path length, as traversed by the simulated point robot, is related to the number of nodes in the final path. As the complexity of the environment increases, more nodes are required to find a path as more obstacles potentially block the most direct route. At lower maximum index level, the final path length is generally shorter as the number of nodes in the final path is lower. However, at these levels solutions are not guaranteed. The LoST algorithm also develops shorter paths as nodes are both placed directly beyond obstacles and intelligently selected based on weighting criteria as opposed to the RRT algorithm's random seeding.

Similarly, as the environmental complexity increases so too does the total distance travelled increase as more nodes are searched from by the simulated point robot. Although lower maximum index levels do not necessarily equate to a lower total distance travelled. The lowest total distance travelled occurs at each environment's optimal index level. This is more pronounced in the 3D environments. If the maximum index level is too low the simulated robot may terminate searches along branches of the LoS tree prematurely, while if the maximum index level is too high branches will potentially over-search regions.

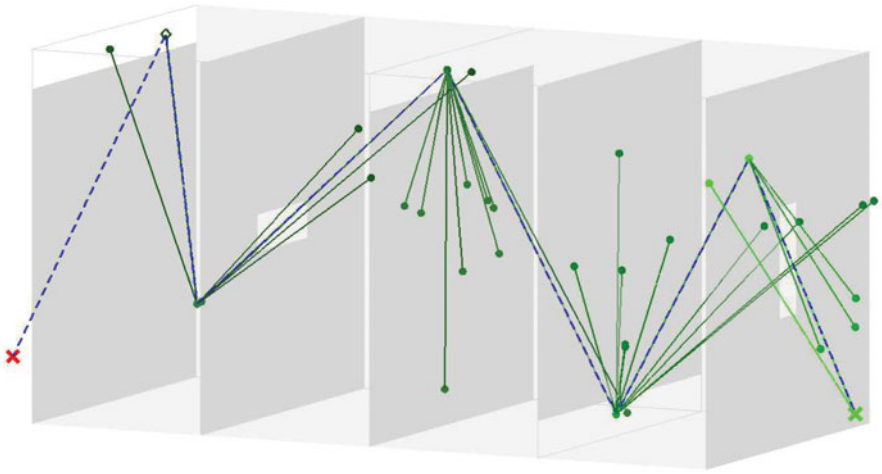
4.3. Verifying the LoST algorithm within the steel bridge tunnel application scenario

This simulation tests the LoST algorithm in a tunnel environment of a steel bridge (Fig. 2). This tunnel environment consists of many sections separated by partition plates. The first partition plate, closest to the start node, has a centred longitudinal slit; the second and third partitions have gaps at the bottom and top respectively; the fourth has a centred crosswise slit; and the final has a gap at the top after which the end node is placed (Fig. 14). The top and front surfaces of the structure are not shown for clarity. The maximum index level for the LoST algorithm has been set to 5. The top and front surfaces of the structure are not shown for clarity.

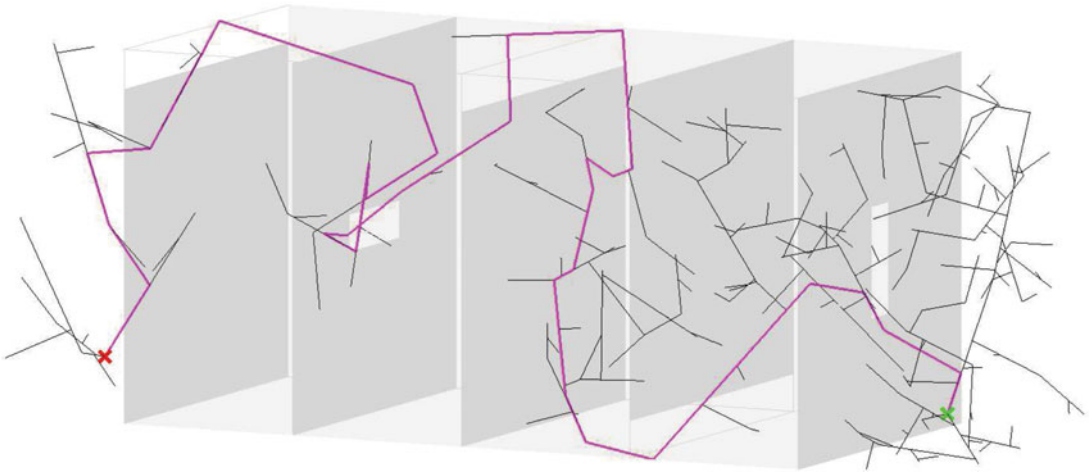
The LoST and RRT algorithms were run 100 times to account for the probabilistic nature of the algorithms. The results are shown in table XIV. Example trees and the final paths generated by the LoST and RRT algorithms are shown in Figs. 14a and 14b respectively.

Both algorithms had a 100% success rate (Table XIV); however, the LoST algorithm managed to find paths with a significantly lower number of nodes. The LoST algorithm always finds a solution along the first LoS tree branch it searches and in five nodes. One node is used to "see" beyond each partition plate which accounts for the five nodes in each search. The LoST algorithm is well suited to find paths between obstacles and in gaps, provided the resolution between rays is significantly small. While these small gaps are beneficial for the LoST algorithm, the RRT struggles to find interconnecting nodes through the obstacles as evident by the total number of nodes in the final path and found in total. In comparison, the RRT algorithm required on average 446% more nodes in the

https://doi.org/10.1017/S0263574716000047 Published online by Cambridge University Press



(a) LoST path through a simulated steel bridge tunnel environment



(b) RRT path through a simulated steel bridge tunnel environment

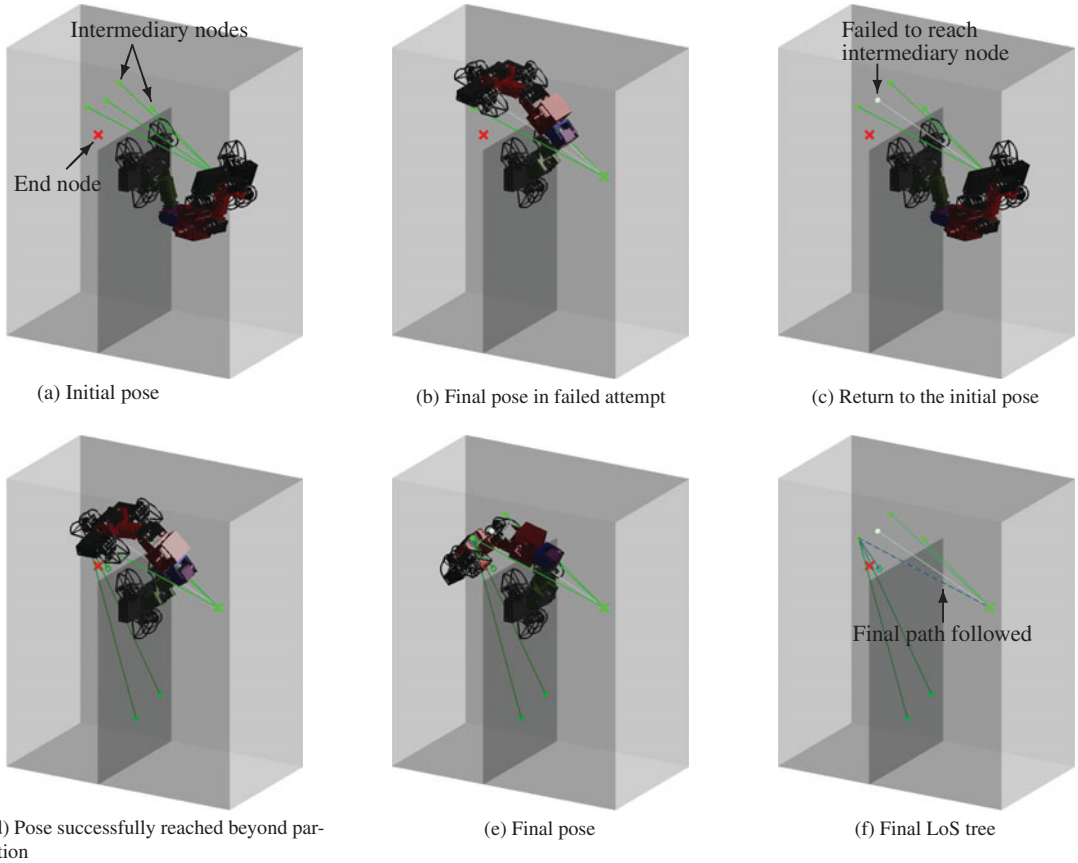
Fig. 14. Simulated steel bridge tunnel environment with example (a) LoST (b) RRT paths shown. The start and end nodes are green and red crosses. The LoS tree connectivity and intermediate nodes are shown as green lines and dots with the final path the blue dotted line. The RRT tree is black with the final path magenta.

final path and 2157% more in total. The LoST algorithm's final path length is shorter as intermediate nodes are positioned just beyond the partitions, as opposed to the RRT algorithm which randomly seeds nodes, preventing direct paths and increasing the final length travelled. As the total distance travelled is the same as the final path length, the LoST algorithm was able to find solutions without needing to backtrack along its path.

4.4. Robot simulations in the steel bridge tunnel application scenario

The presented approach was used to develop a path for a 7DOF inchworm robot within an unknown environment derived from the steel bridge (Fig. 2b). The environment is a partition plate, a solid plate with access through a gap at the top (Fig. 15a), and is used to verify the functionality of the approach. A major difficulty arises due to the size of the force field surrounding the end-effector in comparison to the size of the gap. In the worst case scenario, the relative size of the force field is 116% the size of the gap which requires the end-effector to be orientated correctly to bypass the gap. The inchworm robot's base is placed on the partition plate itself with the end node positioned on the opposing side. To prevent waypoints from being generated within the base, a bounding box was used to encompass the base and act as an obstacle. A simulated Microsoft Kinect sensor is used to perform the necessary ray cast operations.

https://doi.org/10.1017/S0263574716000047 Published online by Cambridge University Press



(a) Initial pose (b) Final pose in failed attempt (c) Return to the initial pose

(d) Pose successfully reached beyond partition (e) Final pose (f) Final LoS tree

Fig. 15. The simulated partition plate environment used to develop a collision free path for an inchworm robot. From (a) the initial pose the robot attempts to move to the end node but fails (b) and subsequently returns to the initial pose (c). Eventually (d) a pose is reached beyond the partition from where (e) the robot moves to the end node. In (f) the final LoS tree shown as a blue dotted line.

First, the region around the inchworm robot's initial pose is scanned with common regions and intermediate nodes determined (Fig. 15a). Scan data is shown in Fig. 16. The inchworm robot attempts to find a smooth, collision-free path to the highest weighted waypoint which, in this case, is an intermediate node. However, the 3D-F^2 algorithm enters a local minima and fails to find a successful path (Fig. 15b). The inchworm robot backtracks to the pose it started at when beginning to move to the waypoint, which is the start pose, and the LoST algorithm is re-invoked to find the next best waypoint. Potentially the inchworm robot would need to backtrack further to the pose corresponding to the last intersecting node between the branches of the old and new waypoints to prevent the 3D-F^2 algorithm from moving along non-LoS paths. However, as the inchworm robot is already at the last intersecting node further backtracking is not necessary (Fig. 15c). From this pose, the 3D-F^2 algorithm successfully determines a path to the new best waypoint (Fig. 15d). The LoST algorithm is updated from this pose and the end node found. The 3D-F^2 algorithm again successfully develops a path for the inchworm robot to follow to the end node (Fig. 15e). The final LoS tree is shown in Fig. 15f. Without the LoST algorithm providing waypoints for the inchworm robot to follow, the 3D-F^2 algorithm alone would not have been able to successful find a path through the environment.

4.5. Robot experiment in an application scenario environment

The inchworm robot is manoeuvred through an environment derived from the steel bridge by a path generated with the approach. From the initial pose (Fig. 17a), intermediate nodes and common regions are determined by searching around the initial pose (Fig. 17b). An intermediate node is found as the next best waypoint to move towards. However, this waypoint cannot be reached as a local minima region is entered (Fig. 17c) and the inchworm robot is returned to the initial pose (Fig. 17d). The next

https://doi.org/10.1017/S0263574716000047 Published online by Cambridge University Press

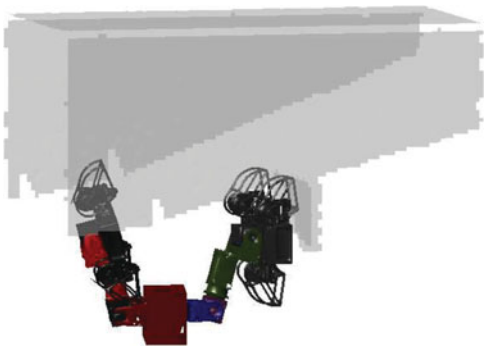


Fig. 16. Simulation scan data taken from the initial pose.

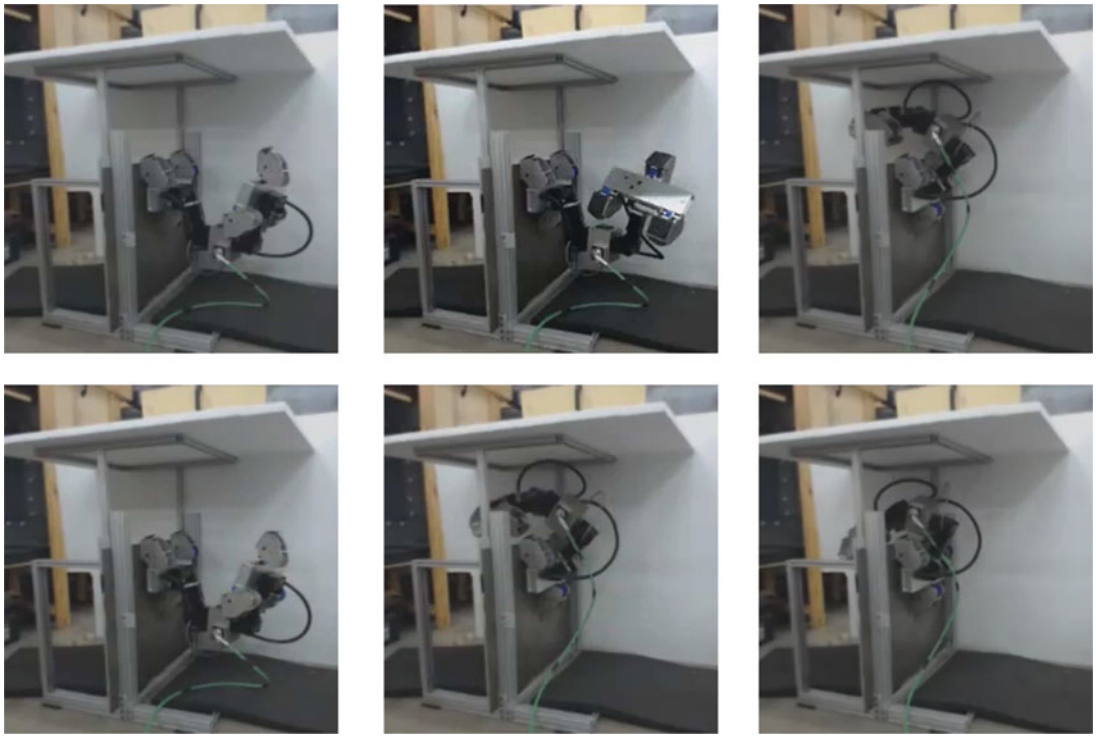


Fig. 17. The inchworm robot successfully following a path generated using the presented approach through a derived application scenario environment. (a) Initial pose. (b) Searching around the initial pose. (c) Local minima pose. (d) Returning to initial pose. (e) Path followed to the supplied intermediate node. (f) Final pose.

best waypoint is then determined and is reached by the inchworm robot successfully (Fig. 17e). From this waypoint, the end node is found and the inchworm robot successfully follows the path generated by the approach through the environment (Fig. 17f). The encoder readings from each joint and the distance to the end node are both shown in Fig. 18. It can be seen that after detecting the local minima the robot returns to the initial pose before successfully moving through the environment.

5. Discussion on the Drawback with the Line-of-Sight Tree Algorithm

The major issue with the LoST algorithm, which is common for many planners, is the difficulty in finding paths between close obstacles. Figures 19a and 19b show two environmental situations where this may occur. Two factors contribute to this issue. First, if the resolution between each ray is too low, rays will not penetrate small gaps preventing the LoST algorithm from detecting them. Second, if the threshold which dictates if two rays are discontinuous is too low, small gaps will not be detected.

https://doi.org/10.1017/S0263574716000047 Published online by Cambridge University Press

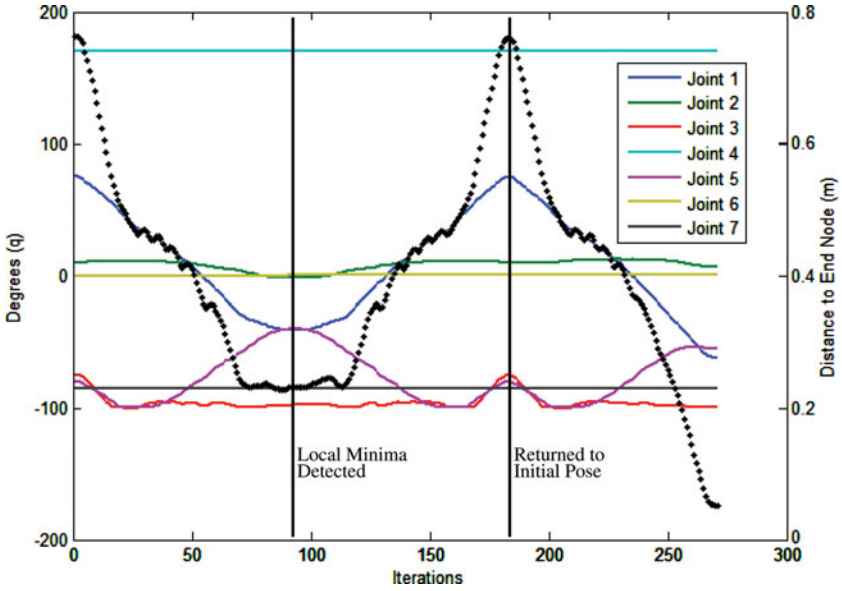


Fig. 18. Graph of joint readings with distance to end node superimposed (dotted line).

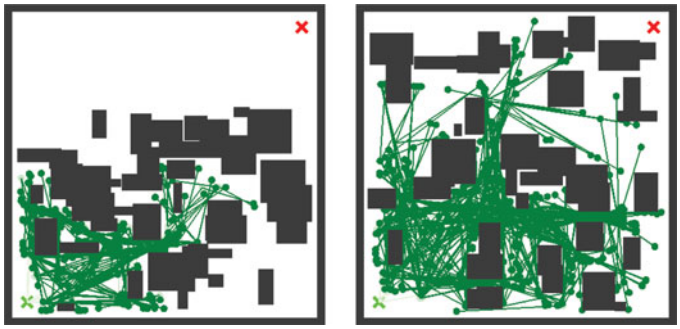


Fig. 19. Two example environments with unreachable end nodes due to undetectable gaps.

6. Conclusion

An approach was presented which uses sensor information to navigate an inchworm robot through complex environments. This is achieved in a similar manner to how a person perceives a scene whereby their gaze tends towards information-rich regions, such as edges of objects, which then leads their gaze to other regions. The inchworm robot "looks" around its current pose to determine obstacle edges to move towards and continue searching for the end node. Results showed that the LoST algorithm is capable of performing path planning for navigation of a robot through complex environments by determining waypoints in Cartesian space.

The LoST algorithm was also presented and compared to an RRT algorithm. In all cases, the LoST algorithm generated paths with fewer intermediate nodes, and a shorter final path length than the tested RRT algorithm. Like any planner, the LoST algorithm is dependent on the environment with variables needing to be tuned accordingly. Future work on the LoST algorithm includes methods to end branches of the LoS tree when subsequent searches will not provide significant new information, and filter out new nodes if they are within close proximity to existing nodes within the tree. These improvements would reduce the number of collision checks and allow a solution to be found quicker. Future work also includes development of methods for handling uncertainties in sensor readings and in robot localisation caused by sag of the inchworm robot due to its design and mechanical structure.

Acknowledgements

This work is supported by the NSW Roads and Maritime Services and the Centre for Autonomous Systems (CAS) at the University of Technology Sydney.

https://doi.org/10.1017/S0263574716000047 Published online by Cambridge University Press

References

1. J. Amanatides and A. Woo, "A Fast Voxel Traversal Algorithm for Ray Tracing," *Proceedings of the Eurographics*, Amsterdam, Netherlands (1987) pp. 3–10.
2. J. Borenstein and Y. Koren, "Real-time obstacle avoidance for fast mobile robots," *IEEE Trans. Syst. Man Cybern.* **19**(5), 1179–1187 (1989).
3. N. Buniyamin, N. Sariff, W. Wan Ngah and Z. Mohamad, "Robot global path planning overview and a variation of ant colony system algorithm," *Int. J. Math. Comput. Simul.* **5**(1), 9–16 (2011).
4. H. Choset, "Coverage for robotics: A survey of recent results," *Ann. Math. Artif. Intell.* **31**(1–4), 113–126 (2001).
5. P. Chotiprayanakul, D. Liu and G. Dissanayake, "Human-robot-environment interaction interface for robotic Grit-blasting of complex steel bridges," *Autom. Constr.* **27**(0), 11–23 (2012).
6. M. Clifton, G. Paul, N. Kwok, D. K. Liu and D.-L. Wang, "Evaluating Performance of Multiple RRTs," *Proceedings of the IEEE/ASME International Conference Mechatronic and Embedded Systems and Applications*, Beijing, China (2008) pp. 564–569.
7. M. Dorigo, V. Maniezzo and A. Colorni, "Ant system: Optimization by a colony of cooperating agents," *IEEE Trans. Syst. Man Cybern. Part B* **26**(1), 29–41 (1996).
8. N. A. Dung and A. Shimada, "A Path-planning Algorithm for Humanoid Climbing Robot using Kinect Sensor," *Proceedings of the SICE Annual Conference (SICE)*, Japan (2014) pp. 1549–1554.
9. D. Ferguson, N. Kalra and A. Stentz, "Replanning with RRTs," *Proceedings of the IEEE International Conference on Robotics and Automation, ICRA*, Orlando, Florida (2006) pp. 1243–1248.
10. F. Flacco, T. Kröger, A. D. Luca and O. Khatib, "A Depth Space Approach to Human-Robot Collision Avoidance," *Proceedings of the IEEE International Conference on Robotics and Automation*, Saint Paul, MN, USA (2012) pp. 338–345.
11. Y. Fu, M. Ding and C. Zhou, "Phase angle-encoded and quantum-behaved particle swarm optimization applied to three-dimensional route planning for UAV," *IEEE Trans. Syst. Man Cybern. Part A* **42**(2), 511–526 (2012).
12. J. D. Gammell S. S. Srinivasa and T. D. Barfoot, "Informed RRT*: Optimal Incremental Path Planning Focused through an Admissible Ellipsoidal Heuristic," *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2014)*, volume abs/1404.2334, Chicago, Illinois (2014) pp. 2997–3004.
13. S. S. Ge, Q. Zhang, A. T. Abraham and B. Rebsamen, "Simultaneous path planning and topological mapping (SP2ATM) for environment exploration and goal oriented navigation," *Robot. Auton. Syst.* **59**(3-4), 228–242 (2011).
14. S. Haddadin, R. Belder and A. Albu-Schaeffer, "Dynamic Motion Planning for Robots in Partially Unknown Environments," *Proceedings of the 18th IFAC World Congress*, *World Congress*, , International Federation of Automatic Control. Milano, Italy, vol. 18 (2011) pp. 6842–6850.
15. J. M. Henderson "Human gaze control during real-world scene perception," *Trends Cogn. Sci.* **7**(11), 498–504 (2003).
16. A. Jain, M. D. Killpack, A. Edsinger and C. C. Kemp, "Reaching in clutter with whole-arm tactile sensing," *Int. J. Robot. Res.* **32**(4), 458–482 (2013).
17. M. A. K. Jaradat, M. H. Garibeh and E. A. Feilat, "Autonomous mobile robot dynamic motion planning using hybrid fuzzy potential field," *Soft Comput.* **16**(1), 153–164 (2012).
18. N. Kalra, D. Ferguson and A. Stentz, "Incremental reconstruction of generalized Voronoi diagrams on grids," *Robot. Auton. Syst.* **57**(2), 123–128 (2009). Selected papers from 9th International Conference on Intelligent Autonomous Systems (IAS-9).
19. S. Karaman and E. Frazzoli, "Sampling-Based Algorithms for Optimal Motion Planning," *CoRR*, abs/1105.1186:1–76 (2011).
20. J. Kennedy and R. Eberhart, "Particle Swarm Optimization," *Proceedings IEEE International Conference on Neural Networks*, vol. 4, Perth, Western Australia (1995) pp. 1942–1948.
21. O. Khatib, "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots," *Proceedings IEEE International Conference on Robotics and Automation*, vol. 2, St. Louis, Missouri (1985) pp. 500–505.
22. Y. Koren and J. Borenstein, "Potential Field Methods and Their Inherent Limitations for Mobile Robot Navigation," *Proceedings of the IEEE International Conference on Robotics and Automation*, vol. 2, Sacramento, California (1991) pp. 1398–1404.
23. Lepora, N. F., P. Verschure and T. J. Prescott, "The state of the art in biomimetics," *Bioinspiration Biomimetics* **8**(1), 1–11 (2013).
24. Q. Li, L. Wang, B. Chen and Z. Zhou, "An Improved Artificial Potential Field Method for Solving Local Minimum Problem," *Proceedings of the 2nd International Conference on Intelligent Control and Information Processing (ICICIP)*, vol. 1, Harbin, China (2011) pp. 420–424.
25. M. Mujahed, H. Jaddu, D. Fischer and B. Mertsching, "Tangential Closest Gap based (TCG) Reactive Obstacle Avoidance Navigation for Cluttered Environments," *Proceedings of the IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, Linköping, Sweden (2013) pp. 1–6.
26. Murphy, M. P., C. Kute, Y. Mengüç and M. Sitti, "Waalbot II: Adhesion recovery and improved performance of a climbing robot using fibrillar adhesives," *Int. J. Robot. Res.* **30**(1), 118–133 (2011).
27. New South Wales Government (2011). WorkCover.

28. D. N. Nia, H. S. Tang, B. Karasfi, O. R. E. Motlagh and A. C. Kit, "Virtual force field algorithm for a behaviour-based autonomous robot in unknown environments," *Proc. Inst. Mech. Eng. Part I: J. Syst. Control Eng.* **225**(1), 51–62 (2011).
29. N. Nourani-Vatani, M. Bosse, J. Roberts and M. Dunbabin, "Practical Path Planning and Obstacle Avoidance for Autonomous Mowing," *Proceedings of the Australasian Conference of Robotics and Automation*, Auckland, New Zealand (2006) pp. 1–9.
30. D. Pagano, D. Liu and K. Waldron, "A Method for Optimal Design of an Inchworm Climbing Robot," *Proceedings of the IEEE International Conference on Robotics and Biomimetics (ROBIO)*, Guangzhou, China (2012) pp. 1293–1298.
31. G. Paul, D. Liu and N. Kirchner, "An Algorithm for Surface Growing from Laser Scan Generated Point Clouds," *Robotic Welding, Intelligence and Automation*, Springer, New York (2007) pp. 481–491.
32. G. Paul, S. Webb, D. Liu and G. Dissanayake, "Autonomous robot manipulator-based exploration and mapping system for bridge maintenance," *Robot. Auton. Syst.* **59**(7–8), 543–554 (2011).
33. D. W. Payton, M. J. Daily, B. Hoff, M. D. Howard and C. L. Lee, "Pheromone Robotics," *Proceedings of the Intelligent Systems and Smart Manufacturing*, International Society for Optics and Photonics, Boston, Massachusetts (2001) pp. 67–75.
34. A. Qureshi, K. Iqbal, S. Qamar, F. Islam, Y. Ayaz and N. Muhammad, "Potential Guided Directional-RRT* for Accelerated Motion Planning in Cluttered Environments," *Proceedings of the IEEE International Conference on Mechatronics and Automation (ICMA)*, Takamatsu, Kagawa, Japan (2013) pp. 519–524.
35. A. T. Rashid, A. A. Ali, M. Frasca and L. Fortuna, "Path planning with obstacle avoidance based on visibility binary tree algorithm," *Robot. Auton. Syst.* **61**(12), 1440–1449 (2013).
36. P. Romey, A. Nhan, K. Williams and M. Dunn, "Sydney Harbour Bridge Conservation Management Plan 2007," Technical report, Roads and Traffic Authority.
37. D. Schmidt and K. Berns, "Climbing robots for maintenance and inspections of vertical structures: A survey of design aspects and technologies," *Robot. Auton. Syst.* **61**(12), 1288–1305 (2013).
38. Z. Shiller and S. Dubowsky, "On computing the Global time-optimal motions of robotic manipulators in the presence of obstacles," *IEEE Trans. Robot. Autom.* **7**(6), 785–797 (1991).
39. M. Shiomi, F. Zanlungo, K. Hayashi and T. Kanda, "Towards a socially acceptable collision avoidance for a mobile robot navigating among pedestrians using a pedestrian model," *Int. J. Soc. Robot.* **6**(3), 443–455 (2014).
40. A. Sintov, T. Avramovich and A. Shapiro, "Design and motion planning of an autonomous climbing robot with claws," *Robot. Auton. Syst.* **59**(11), 1008–1019 (2011).
41. G. Tanzmeister, M. Friedl, D. Wollherr and M. Buss, "Path Planning on Grid Maps with Unknown Goal Poses," *Proceedings of the 16th International IEEE Conference on Intelligent Transportation Systems-(ITSC)*, *IEEE*, The Hague, Netherlands (2013) pp. 430–435.
42. D. Wang, N. M. Kwok, D. K. Liu, H. Lau and G. Dissanayake, "PSO-Tuned F2 Method for Multi-Robot Navigation," *Proceedings of the IEEERSJ International Conference on Intelligent Robots and Systems IROS*, San Diego, California (2007) pp. 3765–3770.
43. P. Ward, G. Paul, P. Quin, D. Pagano, C.-H. Yang, D. Liu, K. Waldron, G. Dissanayake, P. Brooks, P. Mann, W. Kaluarachchi, P. Manamperi and L. Matkovic, "Climbing Robot For Steel Bridge Inspection: Design Challenges," *Proceedings of the 9th Austroads Bridge Conference*, Sydney, New South Wales (2014) pp. 1–12.
44. S. Wen, W. Zheng, J. Zhu, X. Li and S. Chen, "Elman fuzzy adaptive control for obstacle avoidance of mobile robots using hybrid force/position incorporation," *IEEE Trans. Syst. Man Cybern. Part C: Appl. Rev.* **42**(4), 603–608 (2012).