A Parallel Approach to Navigation in Cities using Reconfigurable Mesh

Hatem M. El-Boghdadi[1,2] **and Fazal Noor**[1]
helboghdadi@iu.edu.sa mfnoor@iu.edu.sa
[1]Faculty of Computer & Information Systems, Islamic University of Madinah, Saudi Arabia
[2]*Computer Engineering Department, Ca4iro University, Egypt*

Summary
The subject of navigation has drawn a large interest in the last few years. Navigation problem (or path planning) finds the path between two points, source location and destination location. In smart cities, solving navigation problem is essential to all residents and visitors of such cities to guide them to move easily between locations. Also, the navigation problem is very important in case of moving robots that move around the city or part of it to get some certain tasks done such as delivering packages, delivering food, etc. In either case, solution to the navigation is essential.
The core to navigation systems is the navigation algorithms they employ. Navigation algorithms can be classified into navigation algorithms that depend on maps and navigation without the use of maps. The map contains all available routes and its directions. In this proposal, we consider the first class.
In this paper, we are interested in getting path planning solutions very fast. In doing so, we employ a parallel platform, Reconfigurable mesh (R-Mesh), to compute the path from source location to destination location. R-Mesh is a parallel platform that has very fast solutions to many problems and can be deployed in moving vehicles and moving robots.
This paper presents two algorithms for path planning. The first assumes maps with linear streets. The second considers maps with branching streets. In both algorithms, the quality of the path is evaluated in terms of the length of the path and the number of turns in the path.
Keywords: *Reconfigurable Mesh, path planning, parallel algorithms*

.1. Introduction

The navigation problem is one of the important problems due to its wide range of applications. The navigation problem, also known as path planning problem, is to find the path between two locations, source location and destination location. In smart cities, moving robots (or delivery robots) could be used in closed areas, or even certain geographical area of the city for delivering packages, delivering food, etc. These moving robots (see Figure 1 for an example) need to move between two locations repeatedly and need to solve the path planning problem very fast. Moving robots is a very common problem in robotics research [12]. Also, all residents and visitors of smart cities need navigation systems to guide them to move easily among different locations. In either case, solutions to the navigation problem are essential.

Manuscript received April 5, 2021
Manuscript revised April 20, 2021
https://doi.org/10.22937/IJCSNS.2021.21.4.1

The residents of the city of Madinah are estimated to be over 1.1 million people. It is expected that in 2040, Madinah visitors will reach around 8.5 million Umrah. In such big city, navigation systems are important. This work could be a step to take the city of Madinah to the era of smart cities. The core to navigation systems is the navigation algorithms they employ. Many algorithms exist with different variations and techniques.

One classification of navigation algorithms is dependent on its type of inputs; navigation algorithms that depend on maps and algorithms that depend on visual observations and without the use of maps. The first class involves building explicit maps or uses pre-prepared maps and then planning the route through these maps [11]. The map contains all available routes and its directions and could be for the whole city or certain geographical area of the city (see Figure 2 for an example of a map).

Figure 1. Example for a delivery robot

Figure 2. A map of a certain geographical area

The second class tries to navigate within environments without maps. Instead, it uses visual observations and apply machine learning techniques [22] to navigate. In this paper, we are interested in the first class that achieve navigation with the use of maps. We are also interested in achieving the path planning solutions very fast. In doing so, we plan to employ a parallel platform, Reconfigurable Mesh (R-Mesh), to compute the path from source location to destination location.

Reconfiguration was proven to be a powerful computing paradigm, capable of providing fast solutions to a range of problems. Platforms such as the R-Mesh [2,3] (shown in Figure 3) have the ability to change the interconnection between processors dynamically at every step of the computation to allow efficient communication as well as to perform computation faster than conventional non-reconfigurable platforms.

A 2D R-Mesh is an array of processors that are connected with fixed connections between each two neighboring processors. Also, each processor has internal connections that can be dynamically reconfigured. This allows altering the interconnection among processors very fast, possibly at each step of the computation.

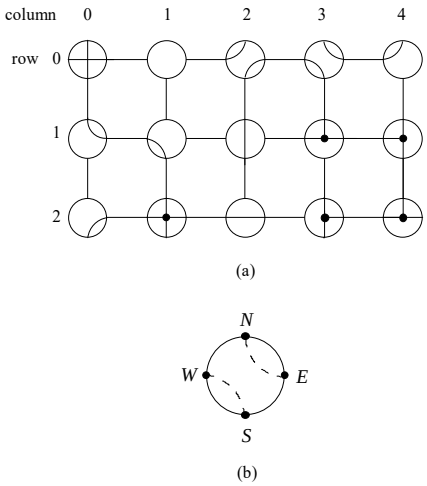

Figure 3. (a) Example of buses in a 3 × 5 R-Mesh (b) Processor ports

Each processor can independently set (partition) its ports to connect certain ports together at each step of the computation. For example, in Figure 3(a) the top left processor connects its N port to its S port, and its E port to its W port. The corresponding partition is denoted by $\{\overline{NS}, \overline{EW}\}$. Figure 3(a) shows the fifteen possible port partitions of the R-Mesh.

A certain setting for a port partition to each R-Mesh processor is called a configuration (see Figure 3(b) for a possible configuration). The port partitions along with the underlying mesh connections between neighboring processors form buses connecting processors (see Figure 3(a) for buses formed in a certain step).

At each step of an R-Mesh algorithm [2,3], a processor could perform the following actions: (1) configure (partition) its ports, (2) read from or write to its ports, and (3) perform a local computation. An R-Mesh could permit concurrent reads and writes.

Researchers have used R-Mesh to design fast and efficient algorithms in image processing, computer vision, arithmetic problems, graph problems, etc. [2,3].

In this paper, we consider the design and implementation of parallel algorithms for the navigation problem on R-Mesh. We first consider maps with linear streets, *i.e.* streets with no branches.

Then we consider streets with branches. In both cases, we propose a parallel approach to solve the navigation problem using the reconfigurable mesh with the use of maps. The proposed algorithms are analyzed with respect of the running time. Finally, we consider the quality of paths in terms of its length and the number of turns in the path.

The next section presents the related work. In section 3, we show the overall system architecture and the methodology followed in this paper. Section 4 presents the path planning algorithm for maps with linear streets. Section 5 considers maps with branching streets. Finally, in section 6, we make some concluding remarks.

2. Related Work

The navigation problem or path planning problem was proved to be very important in many applications. This operation is an essential operation in many navigation systems [21,22] needed in smart cities. Moving robots, moving vehicles, indoor navigation and maze solving [17,18] are examples of such applications.

The navigation problem finds a path from a source location to a destination location within an environment. The core to navigation systems is the navigation algorithms they employ. Two main tracks were followed: the first track solves the navigation problem using maps and the second track solves the problem using visual observations and without the use of maps. In this proposal, we follow the first track.

Using visual inputs has been shown to have some success in some applications [15]. RatSLAM demonstrates localization and path planning over long distances using a biologically-inspired architecture [21]. This method used supervised learning approach. Other methods were proposed that rely also on supervised training with ground truth labels [13,14, 15, 16].

Deep reinforcement learning (RL) was used to learn an agent navigating in indoor buildings [20,21], games [17] and mazes [18]. However, the success was shown for simulated environments more than real applications. Also, the use of RL technique depends very much on environment sensitivity. Piotr Mirowski *et al*. [22] proposed an approach to navigate in a city without a map. The paper presents an end-to-end deep RL approach that can be applied on a city scale. They show that an agent can learn to navigate in multiple cities and to traverse to target destinations that are far away.

The first track solves navigation problem that depends on using maps for the environment. The maps are constructed in the exploration phase [11,12] and then used in the path planning phase to navigate. The map contains all available routes and its directions. This proposal follows this approach and assumes that map is already built in the exploration phase.

For city maps, these maps contain the available paths to follow as well as direction of each path. The map could be for the whole city or certain geographical area in the city.

Models such as the reconfigurable mesh (R-Mesh) [2,3] (shown in Figure 3) have the ability to change the interconnection between processors at every step of the computation to allow efficient communication as well as performing computation faster than conventional non-reconfigurable models. A 2D R-Mesh is an array of processors with fixed external connections between each two neighboring processors. Also, it has dynamically reconfigurable internal connections within each processor. This allows altering the interconnection among processors very fast, possibly at each step of the computation.

Reconfigurable platforms such as R-Mesh [2,3] were used in literature to propose robot path planning algorithms that aim to plan a path for a moving robot from a source location to a target location with the existence of static or dynamic obstacles. The main input to the algorithms is an image (map) with obstacles. In this paper we consider maps with available paths and direction of each path to accommodate the navigation problem within cities. Adding these restrictions to the map would add more complexity to the algorithm.

Restrictions on the robot movement include the robot shape and type of movement; for example, translational or rotational. The R-Mesh was shown to handle these kinds of problems very fast [2,3].

The proposed techniques either use the original map (image) or first an algorithm to generate the configuration space. The configuration space is a slightly different image for the robot and the obstacles than the original image. This greatly simplifies the design and analysis of the algorithm [8].

Tzionas *et al*. presented a parallel algorithm for collision-free path planning [9] for diamond-shaped robot. The configuration space was computed optimally on hypercube in [10]. The algorithm was shown to be optimal where it requires $O(\log n)$ time for an n x n image by using n x n Mesh of processors. D. Wang [7] solved reachability problem in one dimensional space efficiently. Dehne *et al*. [8] presented a systolic algorithm for computing the configuration space for obstacles in a plane for a rectilinear convex robot. The algorithm requires $O(n)$ time for an n x n image on an n x n mesh of processors.

H.C. Lee [5] considered the maze-routing problem. R-Mesh was shown to be suitable for developing efficient and fast algorithms. The path planning problem can benefit from the maze-routing problem.

D. Wang [4] used the 2D R-Mesh to compute a collision-free path. The authors used disjoint convex or concave polygons as obstacles. The algorithm uses $O(k)$ time to compute a path from a source to a destination while avoiding all obstacles in the environment, where N is the total number of processors (pixels) and k is the number of obstacles. However, the algorithm requires $O(\log 2N)$ preprocessing time for the given obstacle image.

The authors in [5] proposed $O(1)$ time algorithm to find a collision free path in the existence of obstacles. Also, the authors proposed a measure for the quality of the path that depends on the number of bends in the path.

All the above algorithms assume an image with obstacles. Thus, it is available to navigate and avoid obstacles. In this work we assume maps with paths and each path has its own direction to accommodate city navigation (see Figure 4).

3. System Architecture

Figure 5 shows the overall proposed system. The main platform is the R-Mesh where it can accept different map areas and generates in parallel the navigation path from a source to destination.; maps with linear streets and maps with branching streets.

Figure 4. Example of a map

In this work, we start surveying the current techniques that handle the navigation problem. After analyzing the current techniques, we used the R-Mesh platform to solve the navigation problem. As mentioned before, one very important advantage is the speed by which we achieve the path from source location to destination location. The input to the algorithm is a map with all routs and its directions. We plan to look at the speed of the solution as well as the quality of the generated solution. We consider two types of maps; maps with linear streets and maps with branching streets.

4. Algorithms for linear streets

In this section, we describe an algorithm to solve the path planning problem using maps with linear streets. In other words, streets that don't have branches. We first start by presenting some known operations on R-Mesh that will be used later.

4.1 Basic Operations

In this section we show some operations on RMesh that will be used in the proposed algorithms.

• Broadcasting Data
Sending information on the buses require constant time and hence broadcasting data to all processors. This can be done by having all processors partition their ports as NSEW. If any processor writes data to one of its ports, data will reach all processors in $O(1)$ time [3].

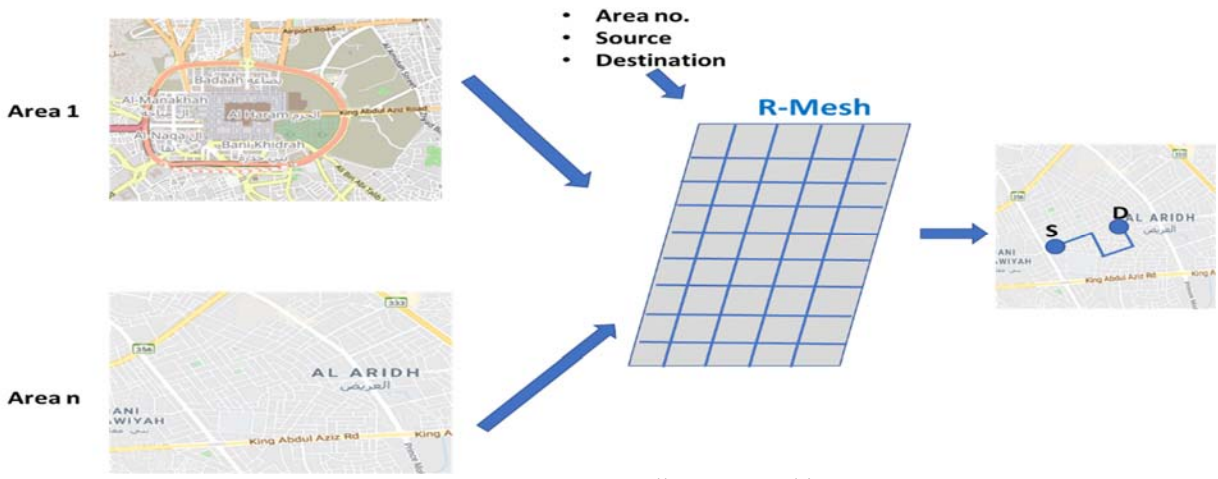

Figure 5. Overall system Architecture

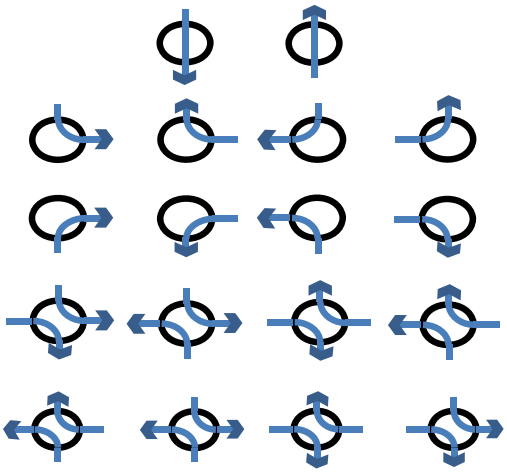

Figure 6. Directions of linear streets through a node

• Counting N bits
Counting N bits on an N x N R-Mesh requires $O(1)$ [3].

Now we describe an algorithm to solve the path planning problem using maps with linear streets. To simplify the problem, the image of map and streets are digitized and stored in the R-Mesh, with one processor holding one pixel of the image. Also, the direction information of streets is stored in each processor. Consequently, each processor knows whether a street is passing through it or not and also the direction of the passing street if any. Since we consider linear streets, any processor could have only at most one street with directions that are shown in Figure 6. The linear street passing through a processor could have one of 18 directions as shown in Figure 6.

Algorithm1

1- Let *Src* node and *Dst* node broadcast their IDs to all processors.
2- Each processor based on local data configure its ports according to streets passing through them.
3- Let *Src* node and *Dst* node cut all buses passing through them.
4- The *Src* node writes its ID on one of the passing outgoing streets at port, *x*.
5- Let all the processors read its ports in sequence.
6- If the *Dst* node reads *Src* ID from one of its ports, *y*, then *Dst* node concludes that there is a route from *Src*(*x*) to *Dst*(*y*).
7- Let the *Dst* node write its ID to same port *y*.
8- Let all the processors read its ports in sequence.
9- If *Src* node reads *Dst* ID from one of its ports, *x*, then there is a route from the *Src* node to *Dst* node starting from port *x* in *Src* node to node *y* in *Dst* node.
10- If a processor reads *Src* ID in Step 5 and *Dst* ID in Step 8, then this processor is part of this route from *Src*(*x*) to *Dst*(*y*). Let all such processor record the route number, *R*, it belongs to.
11- Repeat Step 4 to 10 and let *Src* node to write its ID to another port than *x* (i.e. Steps 4 to10 are repeated two times at most).
12- For each found route, let each processor belong to the route cut the bus and exchange ID with neighboring nodes along the route.

Now we describe Algorithm 1 in details. Step 1 broadcasts the *Src* ID and *Dst* ID to all nodes in RMesh to notify all processors with *Src* node and *Dst* node. In Step 2, all processors configure their ports according to streets passing through it. This allows to send information along the streets according to the actual map. In Step 3, *Src* node and *Dst* node cut all buses passing through them. This will send information along all processors between *Src* node and *Dst* node if they are part of a route.
Since all streets are linear, then at most there could be four different routes from *Src* node to *Dst* node. To check whether there is a route from *Src* to *Dst* or not, *Src* node writes *Src* ID on each port that has an outgoing street (Let the currently written to port is *Src*(*x*)). All processors read their ports in sequence in Step 5. If a processor reads Src ID, then it could be a part of a route from *Src* to *Dst*.
If *Dst* node read the *Src* ID at one of its ports (say *Dst*(*y*)), then there is a route from *Src*(*x*) to *Dst*(*y*). In Step 7, *Dst* node writes its ID on *Dst*(*y*). This will reach *Src*(*x*). All processors read its ports in Step 8.
If a processor read *Src* ID in Step 5 and *Dst* ID in Step 8, then this this processor is part of this route from *Src*(*x*) to *Dst*(*y*). Let each such processor record the route number it belongs to.
In Step 11, the above steps are repeated with the *Src* node writing its ID to a different port than *x*.
Finally, in Step 12, to reach from *Src* node to *Dst* node, for each found route, let each processor belong to the route cut the bus and exchange ID with neighboring nodes along the route. Now each processor knows its successor and predecessor.

Time Analysis

In this section we show that the algorithm requires constant time. Step 1 broadcast the *Src* ID and *Dst* ID to all processors which require $O(1)$ time on RMesh. Steps 2 and 3 perform some port partitioning based on local data which requires constant time. In Steps 4 and 5, various processors either write to a port or read from a port. Both operations runs in $O(1)$ time. In Steps 7 and 8 and 10, various processors either write to a port or read from a port. Both operations run in $O(1)$ time. In step 11, the above operations are repeated at most four times which requires $O(1)$ time. Cutting the buses in Step 12 requires constant time. Thus, the whole algorithm runs in $O(1)$ time.

Theorem 1 For a map with linear streets, an NxN R-Mesh can compute the route from a Source node to a destination node in $O(1)$ time. ■

Since all the steps of the algorithm requires port partitioning with no branches, then the algorithm can be transferred to run on the linear RMesh (LR Mesh). Only Step 1 requires {NSEW} to broadcast data in constant time. This broadcasting can be done also on LR Mesh in $O(1)$ time if a linear bus is constructed that snakes all the processors. This partitioning also can be done in $O(1)$ time. Thus, we have the following result.

Theorem 2 For a map with linear streets, an NxN LR-Mesh can compute the route from a Source node to a destination node in $O(1)$ time. ■

Length of the route

Here we measure the length of the route from *Src* node to *Dst* node. The length of the route could be measured in the number of processors the route snakes or alternatively it could be measured in the number of turns in the route. We first present a method to measure the length of the route in terms of number of processors.
Since all processors belonging to the route could be in different rows, we can do the following:

1- Form a list of all processors belonging to the route in the same row.
2- Count the number of processors in each row in sequence.

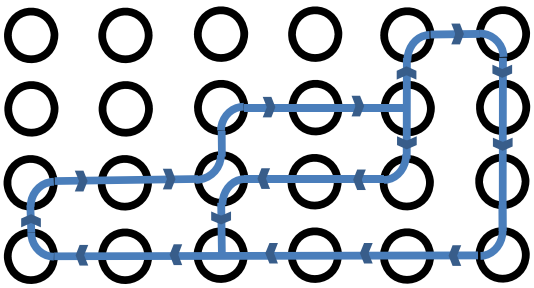

Figure 7. Example of branching streets

Lemma 3 The number of processors belonging to the route from *Src* node to *Dst* node can be computed using N x N R-Mesh in $O(N)$ time.

Proof. Counting the number of processors in each row requires $O(1)$ time. Since the mesh has N rows, then the total time required to count all processors in all rows belonging to the route, in sequence, is $O(N)$. ■

Since each an N x N R-Mesh can compute the number of processors in same row in $O(1)$ time, then the an N x N^2 R-Mesh can compute the total number of processors belonging to the same route in $O(1)$ time. Thus, we have the following result.

Lemma 4 The number of processors belonging to the route from *Src* node to *Dst* node can be computed using N x N^2 R-Mesh (or LR-Mesh) in $O(1)$ time. ■

Number of Turns

We can follow similar procedure as the one followed in measuring the length of the route as follows:

1- Each processor with a turn sets a flag with 1. The rest of processors set the flag with 0.
2- Form a list of at most n flagged processors in each row.
3- Count the number of processors in the list. Thus, we have the following results.

Lemma 5 The number of turns from *Src* node to *Dst* node can be computed using N x N R-Mesh in $O(N)$ time. ■

Lemma 6 The number of turnse from *Src* node to *Dst* node can be computed using N x N^2 R-Mesh (or LR-Mesh) in $O(1)$ time. ■

Now, we consider having streets with branching streets.

5. Algorithms for streets with branches

In this section, we describe an algorithm to solve the path planning problem using maps with branching streets. In other words, streets that have branches. Here we assume that all branches are perpendicular. Figure 7 shows an example of streets with branches. Again, the image of map and streets are digitized and stored in the R-Mesh, with one processor holding one pixel of the image. Also, the direction information of streets is stored in each processor. Consequently, each processor knows whether a street is passing through it or not and also the direction of the passing street if any.

Algorithm2

1- Let *Src* node and *Dst* node broadcast their IDs to all processors.
2- Each processor based on local data configure its ports according to streets passing through them.
3- Let *Src* node and *Dst* node cut all buses passing through them.
4- Let *Src* node writes its ID on one of the passing outgoing streets at port x.
5- Let all the processors read its ports in sequence.
6- If the *Dst* node reads *Src* ID from one of its ports, y, then y node concludes that there is a route from the *Src* node to *Dst* node.
7- Let the *Dst* node write its ID to same port y.
8- Let all the processors read its ports in sequence.
9- If the *Src* node reads *Dst* ID from one of its ports, x, then there is a route from the *Src* node to *Dst* node starting from port x in *Src* node to node y in *Dst* node.
10- If a processor reads *Src* ID in Step 5 and *Dst* ID in Step 8, then this processor **could be** part of this route from *Src* to *Dst*. Let all such processor record the route number, R, it belongs to.
11- (Remove useless routes) If a processor receives *Src* ID and *Dst* ID and this port is not connected, then the processor sends special character on the same port and let all branching processors cut the buses.
12- All processors which receive the character are not part of the route.
13- Each processor with bends or branches cut the bus and exchange IDs.

14- Now starting from *Src* node, the route follows the next processor till reach the *Dst* node.

Now we describe Algorithm 2 in more details. Step 1-8 follow the same lines as Algorithm 1. If a processor read *Src* ID in Step 5 and *Dst* ID in Step 8, then this this processor could be part of this route from *Src*(*x*) to *Dst*(*y*). In addition, some processors could receive both *Src* and *Dst* IDs but these processors and the receiving port is not connected. This processor represents an end point of a certain path. The path from this ending processor to the first branching processor (dead path) should be removed from possible routes from *Src* node to *Dst* node. In Step 11, the ending processor send a special character and let all processors with branch connections cut the bus. The special character will be received by all processor in the dead path. Step 13 lets each processor with branch or a turn cut the bus and exchange IDs with neighboring processors. Now each processor knows its successor and predecessor.

Time Analysis

In this section we show that Algorithm 2 requires constant time. Similar to the analysis of Algorithm 1, each step either reconfigure the ports, send or receive, data or perform local computations. Thus, the whole algorithm runs in $O(1)$ time.

Theorem 7 For a map with branching streets, an *N*x*N* R-Mesh can compute the route from a Source node to a destination node in $O(1)$ time. ■

6. Conclusion

In this paper, we have studied the path planning problem and the use of R-Mesh to solve the problem. We considered the solutions that use maps. These solutions could be used in moving robots within a certain area inside the city. Our first algorithm considered map with linear streets. Our analysis shows that the algorithm runs in $O(1)$ time. We also showed how to compute the length of the route and the number of turns in the path.

Then, we presented an algorithm that solve the path planning problem for maps with branching streets. We also showed that the path can be computed in $O(1)$ time.

Future directions include using other parallel platforms. Also, more restrictions on the maps and streets could be considered.

Acknowledgment

This work is done under the grant received by Deanship of research at Islamic University of Madinah (IUM). We also give special thanks to the administration of IUM for their support in every aspect.

References

[1] Jaja J, An introduction to parallel algorithms. Addison Wesley, Redwood City, CA, USA, 1992.

[2] Bondalapati K, Prasanna VK. Reconfigurable computing: architectures, models and algorithms. Curr Sci 78:828–837, 2009.

[3] R. Vaidyanathan and J. L. Trahan, Dynamic Reconfiguration: Architectures and Algorithms (Kluwer Academic/Plenum Publishers), 2004.

[4] D. Wang, A linear-time algorithm for computing collision-free path on reconfigurable mesh, J. Parallel Comput. 34, 487–496, 2008.

[5] Hatem M. El-Boghdadi. Constant Time Algorithm for Computing a Collision-Free Path on R-Mesh with Path Quality Analysis. Journal of Circuits, Systems, and Computers 24(8): 1550112:1-1550112:20. 2015.

[6] H.-C. Lee, Effecient parallel algorithms on recon¯gurable mesh architectures, Ph.D. Dissertation, University of Missouri-Rolla, 1996. <http://www.mis.yzu.edu.tw/faculty/ hlee/csdiss/>.

[7] D. Wang, Two algorithms for a reachability problem in one-dimensional space, IEEE Trans. Syst., Man, Cybern. 28, 1998.

[8] F. Dehne, A. Hassenklover and J. Sack, Computing the con¯guration space for a robot on a mesh-of-processors, Proc. 1989 ICPP 3, 40–47 1989.

[9] P. Tzionas, A. Thanailakis and P. Tsalides, Collision-free path planning for a diamondshaped robot using two dimensional cellular automata, IEEE Trans. Robot. Automat. 13, 237–250, 1997.

[10] J. Jenq and W. Li, Computing the configuration space for a convex Robot on hypercube multiprocessors, Proc. 7th IEEE Symp. Parallel and Distributed Processing, pp. 160–167, 1995.

[11] Michael Hoy, Alexey S. Matveev and Andrey V. Savkin. Algorithms for collision-free navigation of mobile robots in complex cluttered environments: a survey. Robotica, volume 33, pp. 463–497, 2015.

[12] Otte, M.W. A Survey of Machine Learning Approaches to Robotic Path-Planning. 2009.

[13] Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, KateSaenko, andTrevorDarrell. Long-term recurrent convolutional networks for visual recognition and description. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 2625–2634, 2015.

[14] Mateusz Malinowski, Marcus Rohrbach, and Mario Fritz. Ask your neurons: A deep learning approach to

visual question answering. International Journal of Computer Vision,125(1-3):110– 135, 2017.
[15] Rodrigo F Berriel, Lucas Tabelini Torres, Vinicius B Cardoso, Rânik Guidolini, Claudine Badue, Alberto F De Souza, and Thiago Oliveira-Santos. Heading direction estimation using deep learning with automatic large-scale data acquisition. 2018.
[16] Aditya Khosla, Byoungkwon An An, Joseph J Lim, and Antonio Torralba. Looking beyond the visible scene. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 3710–3717, 2014.
[17] Guillaume Lample and Devendra Singh Chaplot. Playing FPS games with deep reinforcement learning. In Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, 2017.
[18] Piotr Mirowski, Razvan Pascanu, Fabio Viola, Hubert Soyer, Andrew Ballard, Andrea Banino, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, Dharshan Kumaran, and Raia Hadsell. Learning to navigate in complex environments. arXiv preprint arXiv:1611.03673, 2016.
[19] Yi Wu, Yuxin Wu, Georgia Gkioxari, and Yuandong Tian. Building generalizable agents with a realistic and rich 3d environment. arXiv preprint arXiv:1801.02209, 2019.
[20] Yuke Zhu, Roozbeh Mottaghi, Eric Kolve, Joseph J. Lim, Abhinav Gupta, Li Fei-Fei, and Ali Farhadi. Target-driven visual navigation in indoor scenes using deep reinforcement learning. In 2017 IEEE International Conference on Robotics and Automation, ICRA, pages 3357–3364, 2017.
[21] Michael J Milford, Gordon F Wyeth, and David Prasser. Ratslam: a hippocampal model for simultaneous localization and mapping. In Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on, volume 1, pages 403–408. IEEE, 2004.
[22] Piotr Mirowski, Matthew Koichi Grimes, Mateusz Malinowski, Karl Moritz Hermann, Keith Anderson, Denis Teplyashin, Karen Simonyan, Koray Kavukcuoglu, Andrew Zisserman and Raia Hadsell. Learning to Navigate in Cities Without a Map. In 32nd Conference on Neural Information Processing Systems (NeurIPS 2019), Montréal, Canada, 2019
[23] D. Kim,C. Celio, D. Biancolin, J. Bachrach,and K. Asanović, " Evaluation of RISC-VRTL with FPGA-Accelerated Simulation." In First Workshop on Computer Architecture Research with RISC-V. 2017.