Automatic Generation of Waypoint Graphs from Distributed Ceiling-Mounted Smart Cameras for Decentralized Multi-Robot Indoor Navigation

Andrew Felder
University of Arkansas
Fayetteville, AR USA
afelder@uark.edu

Dillon Van Buskirk
University of Arkansas
Fayetteville, AR USA
dtvanbus@uark.edu

Christophe Bobda
University of Florida
Gainesville, Florida, USA
cbobda@ece.ufl.edu

ABSTRACT

This work addresses decentralized coordination of autonomous robots with assistance from overhead map-building and path planning cameras. We propose our solution, Decentralized Indoor Smart Mapping and Hierarchical Navigation (DISCMAHN). We propose an algorithm to generate a waypoint map for each overhead camera's region in real time. Furthermore, we propose a modified A* to perform fully-decentralized path planning. Waypoint generation was simulated to ensure it is both effective and efficient. Path planning was simulated on various, randomized environments to show effectiveness. Our method efficiently handles the cases where other robot navigation methods are otherwise weak and ineffective.

ACM Reference Format:
Andrew Felder, Dillon Van Buskirk, and Christophe Bobda. 2019. Automatic Generation of Waypoint Graphs from Distributed Ceiling-Mounted Smart Cameras for Decentralized Multi-Robot Indoor Navigation. In *13th International Conference on Distributed Smart Cameras (ICDSC 2019), September 9–11, 2019, Trento, Italy*. ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3349801.3349814

1 INTRODUCTION

Indoor robot navigation has become a frequently researched area due to the increase of automation to augment jobs within warehouses and many other areas. This navigation is typically a combination of global planning and local planning, where the ground robots execute the results of the local plans. According to literature, [8], one of the standards for robot navigation is to perform all planning on a central server which distributes instructions to each robot. From there, each robot will execute the path simulataneously since the central server calculates the paths relative to others in order to prevent deadlocks. This method is extremely susceptible to failure due to the heavy load that the central server must process; any downtime suffered by the central server limits or entirely halts all autonomous movement.The other standard for robot navigation is for the ground robots to calculate the paths themselves such as in [1], [3], [6], and [7]. Performing planning directly on the robots puts a heavy load on the robot and requires that they be equipped with unnecessarily powerful hardware. Some work has been done to reduce this load on the robots by employing hierarchical graphs which are used to perform path planning algorithms such as [6] and [11]. Most previous work is designed for static environments. Simultaneous Localization and Mapping (SLAM), [13], is the standard solution to dynamic environments, but SLAM is unaware of the changed environment until a robot visits and maps the area.

In this work we show a system in which a hierarchical graph is used within a decentralized system to allow pathing in dynamic environments. This allows for the reduction of load on the robots that a centralized system provides while maintaining the efficiency of a decentralized system. By removing planning from the robots, it allows the robot to focus on supplemental tasks such as obstacle avoidance or cargo handling. Furthermore, our method can handle any change to the environment and map the changes before any robot has to traverse it.

The rest of this work is organized as follows. In Section 2, we define DISCMAHN and its assumptions. Then, we define the problem that DISCMAHN solves. We propose a solution to that problem in Section 3. In Section 4 and 5, we describe our implementation of our graph generation and path planning methods. We evaluate our methods in Section 6 and conclude in Section 7.

ICDSC 2019, September 9–11, 2019, Trento, Italy

ACM ISBN 978-1-4503-7189-6/19/09…$15.00
https://doi.org/10.1145/3349801.3349814

2 DEFINITIONS AND PROBLEM STATEMENT

Assumptions. For this work, it is assumed that the ground robots have Collision Detection capabilities, allowing them to stop if a collision were incoming. This potential collision could be with another robot or a person. We do not discuss any solutions to deadlocks as it is out of the scope of this work. Some work has been done in this direction in [1], [2], and [3]. We will consider this problem in future work.

Definitions. We define an environment for our system which resembles an industrial storage facility. This environment is dynamic and arbitrary. Our system consists of a set of ceiling-mounted smart cameras and a set of robots.

A set of ceiling-mounted smart cameras view the entire working region of the environment, where an individual camera can see a small subset of the entire environment. Each camera has a working region which has slight overlap with all of its neighbors. These cameras are arranged in a grid in order to simplify this neighboring region overlap. The set of cameras are decentralized such that there is no central server performing any of the processing. For simplicity, each camera is mounted at the same height and each camera's region is the same size. At start-up, each camera performs an initialization phase which calculates a waypoint graph that represents its region. Within the system, there exists a set of robots. After camera initilization, the robots received arbitrary instructions from a human user which tell the robot where to go (a goal which, in an industrial setting, would be where it would pick-up or drop-off cargo). The robots do not move synchronously. This greatly increases the complexity for optimal routing with collision prevention, shortest routes, and deadlock avoidance.

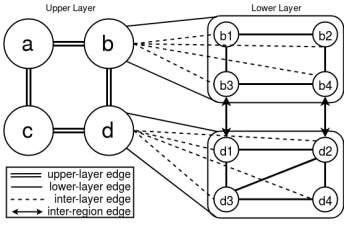

Figure 1: Decentralized Indoor Smart Camera Mapping and Hierarchical Navigation (DISCMAHN)

There exists middleware for communication between all agents in the system - camera-to-camera, robot-to-robot, robot-to-camera, and camera-to-robot. This communication is necessary for decentralization so that each agent can retrieve any information it requires. From here, we can decompose our system into a hierarchical graph where each region within a camera will be a set of lower-level nodes and each camera will be a higher-level node. That is, each camera will be a parent to a set of nodes that exist within the waypoint graph of its region. Furthermore, each camera will have siblings that correspond to the neighboring cameras. This can be seen in Figure 1. Each node on the left side with the labels, $\{a, b, c, d\}$, represents one of the ceiling-mounted smart cameras. These nodes are connected with each of their neighbors and oversee subgraphs. There are two sets of nodes in the lower layer section which represent the region that cameras b and d oversee. Each of the individual nodes in these sets are actual waypoints in the region. There exist inter-region edges, whose weights are considered 0 due to the endpoints overlapping, on the lower layer between regions overseen by neighboring upper layer nodes. Furthermore, Figure 1 shows the relationship between each upper layer node and the nodes within its subgraph denoted by the dashed line. We define our system to be Decentralized Indoor Smart Camera Mapping and Hierarchical Navigation (DISCMAHN).

We can define our upper-level system as an undirected graph, $G = (V, E)$, where V is the set of cameras and where E defines the set of all edges that represent a pair of neighboring cameras. We further define our lower-level system as an undirected graph, $G' = (V', E')$, where V' is the set of all waypoint nodes within each camera and where E' is the set of all edges that represent a pair of waypoint nodes. For each camera, V_i, there exists a unique set of children, $C_i = Y \subset V'$, where Y represents a unique subset of V', such that,

$$\forall\ c_i, c_j \in C \mid c_i \cap c_j = \varnothing \tag{1}$$

Graph G is a parent to graph G' such that

$$\forall\ V_i \in V \mid \exists\ V_i\ \varphi\ (C_i) \tag{2}$$

where φ represents a parent-child relation as shown in Figure 1 above by the dashed lines. A camera, V_i is considered a neighbor of another camera, V_j if and only if there exists an edge between a child of V_i, $c \in C_i$, and a child of V_j, $c \in C_j$.

$$\exists\ (V_i, V_j) \in E(G) \iff \exists\ (c \in C_i, c \in C_j) \in E(G') \tag{3}$$

We define our set of robots as $R = \{R_1, ..., R_k\}$ where the number of robots does not exceed $V(G')$. The set of robots travel along the nodes of the lower-level graph, G'.

Problem Statement. Suppose we have an arbitrary number of robots in a dynamic indoor environment. Also suppose we have a series of decentralized cameras arranged in a grid layout with overlapping field of views between neighboring cameras. Our objective is to dynamically and simultaneously provide guidance to all robots to safely reach their destinations. The objective function considered in this work is a minimum makespan among all robots.

3 PROPOSED SOLUTION

We propose DISCMAHN as an alternative solution to indoor robot navigation. We use an array of fully decentralized ceiling-mounted smart cameras that map out the region. Once the region is mapped, the cameras wait for a robot to

request a path from their region to any other area in the system. Each ceiling-mounted camera is capable of interacting with their neighbors to incrementally build any global path. When a robot requests a path, the source camera performs all the path calculations while requesting needed information from other cameras. When the path is complete, the source camera sends that path to the robot for traversal. At any time, if the environment changes, the ceiling camera overseeing that region detects the change and re-maps the area. Each robot is responsible for driving the path and using the previously assumed method of collision detection.

4 DISCMAHN

Graph Generation

In DISCMAHN, ceiling-mounted smart cameras initialize and capture their region. From there, we decompose the RGB image into a black-and-white subtracted image to symbolize the separation between valid ground and obstacles. Then, we can find waypoints that will serve as nodes in our path planning algorithms in order to speed up the processing of robots in the region.

Image Subtraction. We utilize Mean Shift, a technique for the reduction of multimodal feature space into segmented arbitrary clusters, described in [9], to get operable images of camera regions. We use a histogram to reduce the results of the Mean Shift algorithm into a binary image, as shown in [12]. This method is robust in its ability to decipher ground between any obstacle. However, it requires the assumption that safe ground is the relative majority of the image.

Waypoint Generation. The style of graph that we use for our path planning algorithms is called a waypoint graph. Limited research has been done on automatically generating this specific style of graph. Usually, waypoint graphs are only used in video games and tend to be hard-coded and hand-picked, [10]. However, some research has been done in this area to automatically calculate these graphs, [4], but the results were not tailored enough for our design. Furthermore, [4] used a grid representation of the pathed area, while we rely on a roadmap representation within the child graphs. Due to this, we designed a simple method of extracting a waypoint graph from the binary image resulting from the above mentioned Mean Shift.

This process relies on the calculation of transition regions which are the regions shared by neighboring cameras. We calculate a central point for each transition region on every side which will serve as the start and end point for any path through this region. To find the waypoint graph, we simply perform a path-finding algorithm from each transition region to every other transition region. This would require an algorithm similar to all-pairs shortest path if paths were not bidirectional. In our case, we consider that every path is bidirectional. Upon each path calculation, we save each point we have used. After all paths are done, we can step through the saved points that are sorted by frequency of use. These frequently used points and the endpoints are saved as waypoints.

Algorithm 1 Graph Generation

Require:
Original Image of Region, img
Initially Empty Set of Transition Regions, $transRegion$
Initially Empty Set of Waypoints, $waypoints$
$subtracted$ = MeanShift(img)
$transRegion$ = GetTransRegion($subtracted$)
for each $point_u$ in $transRegion$ **do**
  for each $point_v$ in $transRegion$ **do**
    if $point_u$ is $point_v$ **then**
      continue
    else
      $path_{u,v}$ = PathPlanning($point_u$, $point_v$)
    end if
  end for
end for
for each *pixel cluster* in $subtracted$ **do**
  if *pixel cluster* frequency > threshold **then**
    waypoint += *pixel cluster*
  end if
end for
for each $node_u$ in $waypoint$ **do**
  for each $node_v$ in $waypoint$ **do**
    if ∃ edge($node_u$, $node_v$) in path **then**
      add edge between waypoint nodes, $node_u$, $node_v$
    end if
  end for
end for

The Graph Generation, Algorithm 1, takes in an unaltered image from the ceiling camera. It performs the Mean Shift algorithm described in [12] to get the binary image of the region. Then, we calculate the transition regions of the subtracted image by looping a tile around the border of the image and checking for enough space to fit the respective robot. Then, we loop over each of the central points of each transition region and perform a path finding algorithm between them. In simulation, each path is drawn on the binary image of the region by incrementing the color of the path line drawn by a small amount. Once all paths have been calculated, there are clusters of strong colors corresponding to common intersections. Most automatically generated waypoints tend to place waypoints near corners of obstacles whereas this method will place waypoints centered between any two obstacles. We loop over all the pixels in the image at a given tile size, and, given a certain threshold, we determine if this group of pixels was visited enough to become a waypoint. After all waypoint nodes have been added, we add the edges between them based on the original paths.

Path Planning

To the authors' knowledge, a method of path planning across a decentralized system we have defined does not exist. Similar work has been done for centralized systems in [6] and

[11]. Therefore, we propose a modification to standard A*. As we do not have another method to compare against, we will use a greedy euclidean-based search as comparison. We are assuming that a camera in the upper layer has knowledge of other cameras' positions relative to itself.

Greedy Search. For the greedy method of path planning, we only take into account the euclidean distance between the current camera and the goal camera. Therefore nothing at the child level of the graph is considered. At each camera, we take the neighbor with the lower euclidean distance to the goal camera and set it as the current camera. This would be equivalent to finding the shortest path through a camera in the direction of the goal, and replanning at each camera. This method runs until the goal is found. The path is then rebuilt as described in Algorithm 3

Decentralized-A.* We propose Decentralized A* (D-A*), a modified version of A* to plan a path across a set of decentralized, connected parent nodes. The children of these nodes will be taken into account during the planning process. Work in decentralized path planning has been done in [7], [3]. However, these algorithms run in a decentralized system where a robot or set of robots are self-sufficient. In DISCMAHN, we have a set of parent nodes creating a path through their children which the robots will follow. Because it is decentralized, each parent node will have no knowledge of the internals of its neighbors. Therefore, it will be necessary to use some form of network communication to request necessary information from other parent nodes. This algorithm follows a top-down pathing approach. The majority of the pathing will happen on the parent layer with information such as cost and heuristic coming from the child layer of the requested camera. The cost will be the shortest path across the child layer from the starting edge of the current camera to a point in the overlap region of the neighboring camera. For this algorithm, it is assumed that a pathing algorithm to determine the shortest paths on the child layer of every camera has already run. The primary difference between this and standard A* is the cost function and path rebuilding. This algorithm runs solely on the source camera which forces communication with other cameras for information. The path will be entirely rebuilt on the source camera, and passed along to the robot. Given all of this, we propose the following modifications to standard cost, path reconstruction, and A* algorithms in algorithms 2, 3, and 4 respectively.

Per Algorithm 2, if the source node is not the node we are seeking cost for, it will request cost from C_C for a path from the start point of C_C to a point in the overlap region shared with the neighbor in question. The cost of the path currently consists only of the length. After the goal is found, the path will be rebuilt starting from the goal camera as

Algorithm 2 D-A* Cost

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

Algorithm 3 Rebuild Path

Require:
Goal Camera, C_G
Goal Point, P_G in C_G
1: Let $C = C_G$
2: Let P_F be an initially empty set of edges representing the final path
3: Let P_T be an initially empty set of edges representing an intermediate path
4: **while** C.parent ≠ NULL **do**
5: P_T = *this*.requestPath(C.startPoint, C.endPoint)
6: P_T.append(P_F)
7: $P_F = P_T$
8: P_T.clear()
9: **end while**

described in Algorithm 3. The source camera will request the path between the start and end points from the goal camera, and append it to the final path. The interior of the loop concatenates the most recently requested path onto the front of the final path. Since we are operating on a standard grid, the currently used heuristic is Euclidean distance. However, because this is a modification to standard A*, the heuristic can be changed to fit the situation. The final modification to A* is on line 15 of Algorithm 4. The cost in this implementation is directionally dependent; if the current node is the immediate neighbor of the goal camera, the cost of the path to the final goal point must be considered. Assume a goal camera, C_G has two neighbors, N_1 and N_2. If $cost(N_1, C_G) < cost(N_2, C_G)$, then A* will assign N_1 as the parent of C_G. Assume O_{P1} is the overlap point between N_1 and C_G. Assume O_{P2} is the overlap point between N_2 and C_G. Due to cost being directionally dependent, this could cause a problem when finding the path from the start point of C_G to the goal point. This is because it is possible that

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

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

$$cost(N_1, C_G) + pathCost(O_{P1}, C_G.endpoint) >$$
$$cost(N_2, C_G) + pathCost(O_{P2}, C_G.endpoint)$$

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

Algorithm 4 D-A*

Require:
Start Camera, C_s
Goal Camera, C_g
Start Point P_s in C_s
add C_s to *openSet*
while *openSet* is not empty **do**
  N_C = *openSet.pop*
  if N_C is C_g **then**
    return C_S.rebuild_path()
  end if
  for each *neighbor* of N_C **do**
    if *neighbor* in *closedSet* **then**
      continue
    end if
    if *neighbor* not in *openSet* **then**
      add *neighbor* to *openSet*
    end if
    potentialCost = N_C.costSoFar + c_s.cost(N_C, *neighbor*)
    if *neighbor* is C_G **then**
      ov_point = find_overlap_point(N_C, C_g
      potentialCost = C_S.getPathCost(C_G.startPoint, C_G.endPoint)
    end if
    if *potential_cost* < *neighbor*.costSoFar **then**
      N_C.endPoint = find_overlap_point(N_C, neighbor)
      neighbor.startPoint = N_C.endPoint
      neighbor.parent = N_C
      neighbor.costSoFar = *potentialCost*
      neighbor.costToEnd ← *neighbor*.costSoFar + heuristic()
    end if
  end for
  closedSet.add(N_C)
  openSet.remove(N_C)
end while
return failure

5 RESULTS

We tested graph generation on a large array of images taken from our lab where we attempted to replicate an industrial warehouse storage facility that was scaled down. We ran the graph generation described in Algorithm 1. Simulations were performed on a large set of images using OpenCV in C++ on Ubuntu 16.04LTS. We separated the timing of the Mean Shift and transition region calculation from the actual path finding loop. On average, those two functions completed in 500 milliseconds. This speed is relevant solely because it is relatively quick and will not be noticeable in a decentralized system. Once the binary image is calculated, we use a standard A* as our path finding algorithm at a discretization of 8 pixels per step. We only consider pixels that are a certain buffer distance away from obstacles to ensure that the robot in use has plenty of room to make it through the region.

Waypoint generation averaged 2000 milliseconds per image. Without a waypoint graph, one must calculate more finely discretized paths for each request. The cost of pathing every request quickly exceeds the overhead costs required for the waypoint graph. The runtime of an individual path on a waypoint graph is unparalleled in comparison to pathing on an image and is a trivial result. We show an example case from these simulations which includes the original image captured from a ceiling camera and the resulting waypoint graph in Figure 2. In the event of an environment change, cameras can perform a reduced version of the graph generation algorithm on just the relevant transition regions to update the unsafe waypoint and any connected edges.

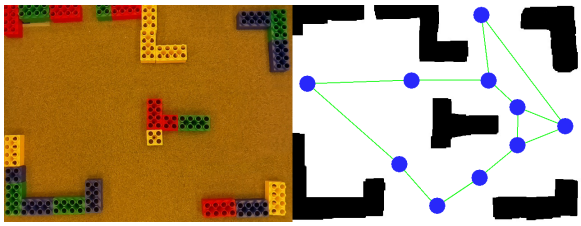

Figure 2: Original Image (Left) and Resulting Waypoint Graph (Right).

To test the effectiveness of the path planning, we simulated an environment of varying sizes. For the top layer, we created a layer of size 5x5, 10x10, 20x20, and 50x50 parent nodes. Each parent node in the layer was given a random number of children between 15 and 20. Random edges were generated among the children of each parent node with the chance of an edge forming between any two child nodes being 25%, 50%, and 75%. This is the connectivity of the graph. In each test run, both pathing algorithms were run on the same graph from the same start point to the same goal point. All of the values in Tables 1, 2, and 3 represent the data collected from D-A* normalized by the values collected from Greedy. These values represent the normalized mean over 100 tests for each combination of the above grid sizes and connectivity rates.

In Table 1, it can be seen that as the connectivity rate increases, the benefit of using D-A* diminishes. This is expected because as the graph becomes closer to fully connected, Greedy will be more likely to find an optimal path. It can also be seen that as the size of the grid increases, the benefit of using D-A* increases. This is also expected because as the size of the graph increases, it is more likely that a shorter, less direct path exists. Table 2 shows that connectivity has no real effect on the amount of requests needed to create a path across a constant grid size. This is because the connectivity rate is local to each parent node's child graph. Therefore, even if every child was fully connected, the parent layer, where communication takes place, connectivity does not change. On the other hand, the number of requests greatly increases as the size of the parent layer increases. This is because D-A* will have a higher branching factor and greater depth as the size of the graph being traversed increases, requiring more communication. The results in Table 3 follow the same pattern as Table 2 for the same reasons. The connectivity of the child layer has no effect on the efficiency of D-A* running on the parent layer. The runtime also increases greatly with the size of the parent layer due to having a larger search space. It should be noted that while D-A* has a very high runtime relative to Greedy Search, the

Table 1: Path Length Results Normalized to Greedy

Path Length			
	25%	50%	75%
5x5	0.888625	0.902511	0.954489
10x10	0.854601	0.871142	0.894761
20x20	0.837341	0.856450	0.880150
50x50	0.810573	0.848751	0.855922

Table 2: Request Count Normalized to Greedy

Network Requests Made			
	25%	50%	75%
5x5	3.59	3.77	3.28
10x10	6.04	5.72	5.53
20x20	9.67	9.77	9.76
50x50	22.58	22.08	22.43

Table 3: Runtime Normalized to Greedy

Runtime			
	25%	50%	75%
5x5	13.145036	13.174933	13.219170
10x10	23.257459	21.718941	21.236186
20x20	35.641699	37.617155	38.646010
50x50	83.810102	82.483988	83.024822

actual runtime was still under one-tenth of a second on the 50x50 layer on average.

6 CONCLUSION

In this work, we presented DISCMAHN, a system designed to work in a static or dynamic environent without loss of efficiency in navigation. Furthermore, we presented an algorithm to generate a waypoint map given an overhead region image and a modified version of A* to allow for decentralized path planning. Other methods of decentralized robot navigation are more suited for static environments, or, in dynamic environments, require revisiting a changed region. Waypoints are recalculated any time a change within the overhead camera's region is detected. This is more efficient than requiring a robot to manually repath the affected area. Once all waypoints are generated, D-A* can be run on the overhead camera layer to plan a path between any two locations in a decentralized manner. This allows a single path to be generated by the source camera rather than planning a new path within each region. Therefore, it is more likely that the path is optimal as the latter would follow a greedy ideology. Simulation showed that waypoint generation from any arrangement of obstacles is completed in an average of 2000 ms in the test environment. This time is significantly lower than the time needed for a robot to re-map the changed area. D-A* consistently returns shorter paths than the greedy method while increasing runtime and requests made on the network. Due to D-A* planning paths incrementally, a recently changed waypoint graph in a camera's subgraph will not affect the effectiveness of D-A*. As stated in section 2.0.1, we will consider deadlock prevention in future work. We will also use neural networks to process the image and compare speeds with our current waypoint generation methods in future work. Furthermore, we will consider the optimality of our D-A* method in future work. This future work will also include congestion analysis which will factor into pathing.

ACKNOWLEDGMENT

This work was supported in part by the national science foundation (NSF) under Grant 1547934.

REFERENCES

[1] V. Digani, L. Sabattini, C. Secchi, and C. Fantuzzi. "Hierarchical Traffic Control for Partially Decentralized Coordination of Multi AGV Systems in Industrial Environments." *Proceedings of the IEEE International Conference On Robotics & Automation (ICRA), 2014.*, pp.6144-6149.

[2] M. Jager and B. Nebel. "Decentralized collision avoidance, deadlock detection, and deadlock resolution for multiple mobile robots." *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems, 2001.*, vol. 3, 2001, pp. 1213-1219 vol.3.

[3] I. Draganjac, D. Miklić, Z. Kovačić, G. Vasiljevi'c, S. Bogdan. "Decentralized Control of Multi-AGV Systems in Autonomous Warehousing Applications." *Proceedings of the IEEE Transactions on Automation Science and Engineering, 2016.*, vol. 13, no. 4, pp. 1433-1447.

[4] D. Jia, C. Hu, K. Qin, and X.Cui. "Planar Waypoint Generation and Path Finding in Dynamic Environment." *Proceedings of the International Conference on Identification, Information, and Knowledge in the Internet of Things, 2014.*, pp. 206-211.

[5] J. Yu and S. LaValle. "Optimal Multi-Robot Path Planning on Graphs: Complete Algorithms and Effective Heuristics." CoRR, abs/1507.03290, July 2015.

[6] C. Zhong, S. Liu, B. Zhang, Q. Lu, J. Wang, Q. Wu, and F. Gao. "A Fast On-line Global Path Planning Algorithm Based on Regionalized Roadmap for Robot Navigation." IFAC-PapersOnLine, vol. 50, issue 1, pp. 319-324, July 2017.

[7] P. Velagapudi, K. Sycara, and P. Scerri. "Decentralized prioritized planning in large multirobot teams." *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2010.*, pp. 4603-4609.

[8] L.E. Parker. "Path Planning and Motion Coordination in Multiple Mobile Robot Teams." *Encyclopedia of complexity and system science, 2009.*, pp. 5783-5800.

[9] D. Comaniciu, and P. Meer. "Mean Shift: A Robust Approach Toward Feature Space Analysis." *IEEE Transactions on pattern analysis and machine intelligence.*, vol. 24, no. 5, May 2002.

[10] X. Cui, and H. Shi. "A*-based pathfinding in modern computer games." *International Journal of Computer Science and Network Security.*, vol. 11, no. 1, pp. 125-130, 2011.

[11] J.Y. Lee, and W. Yu. "A Coarse-to-Fine Approach for Fast Path Finding for Mobile Robots." *Proceedings from IEEE/RSJ International Conference on Intelligent Robots and Systems, 2009.*, pp. 5414-5419.

[12] F.J. Streit, M.J.H. Pantho, C. Bobda, C. Roullet. "Vision-Based Path Construction and Maintenance for Indoor Guidance of Autonomous Ground Vehicles Based on Collaborative Smart Cameras." *Proceedings of the 10th International Conference on Distributed Smart Camera, 2016.*, pp. 44-49.

[13] M.G. Dissanayake, P. Newman, S. Clark, H.F. Durrant-Whyte, and M. Csorba. "A solution to the simultaneous localization and map building (SLAM) problem." *IEEE Transactions on robotics and automation,*

2001., vol. 17, no. 3, pp.229-241.