Project MAERA
Map Explorer: A Distributed Robotics Application

Akshay Choche and Muthukumaran Chandrasekaran

Department of Computer Science, University of Georgia, Athens, GA 30602
{achoche, mkran}@uga.edu

Abstract. Distributed systems find applications in various fields such as robotics, network security etc. In the field of robotics, we can consider a team of robots working towards a common goal to be a distributed system. A team of robots mapping an area can potentially combine their information to explore a given map more efficiently than a single robot alone. We describe a simple, yet effective approach for distributed map exploration. We test the algorithm on a simulated 2D environment. We show the elegance with which the algorithm is able to localize the position of the explorer robots and how processing tasks are assigned dynamically on the fly.

1 Motivation

For applications such as search and rescue operations, security monitoring, urban reconnaissance and even house cleaning, systems of multiple robots must be able to cooperatively explore and map an environment. For example, consider a hypothetic train-wreck situation. To begin with, human intervention could be dangerous because of the potential risk of safety as the crash is bound to have created unstable structures with a possible risk of collapse. This calls for unmanned exploration of the crash site before human teams can actually carryout their search and rescue operation. A team of surveillance robots could be deployed at multiple entry locations to fetch audio and visual information which can then be used by human rescue teams to plan their operation beforehand. Also, these surveillance robots, often small in size, can navigate through regions where human intervention is physically impossible. Their task is to jointly explore the entire map and quickly with minimal redundancy in the areas explored. In this project, we will simulate a team of robots faced with the task of coordinated exploration. Such problems have been previously studied and researchers claim that autonomous exploration and map merging algorithms to be increasingly robust on single robots. But in order to create a map quickly, multiple robots have to be used and each robot can be allowed to explore only a portion of the environment and the robots' individual partial maps will be then combined to form the complete maps. Hence, researchers are faced with their next challenge; extending these techniques to a team of robots. Some difficult issues, including

limited communication between robots, no assumptions about start locations of the robots, and dynamic assignment of processing tasks, hamper progress in this field of research. In this paper, we will address some of these issues while identifying a distributed approach to the coordinated exploration problem.

2 Related Work

The ability to build a map of an unknown environment is one of the fundamental enabling capabilities for mobile robots. Having an accurate map enables the robot to perform certain tasks like navigation, localization and so forth much faster and more accurately. In the past years a significant amount of research has been devoted to this subject. Previous implementations of mapping and exploration have been done robustly on individual robots using laser range finders and visual information [Yamauchi1998] [Thrun, Burgard, and Fox2000]. But in order to bulid a map quickly and efficiently, use of multiple robots are proven to be more beneficial where each robot is allowed to explore only a portion of the map and their individual efforts are combined to complete the exploration in reduced time. However, coordinating multiple robots to efficiently explore a map is a known hard problem. We seek inspiration from Stefano Carpin et al's work on map merging [Carpin, Birk, and Jucikas2005]. Their work mainly deals with fusing two or more partial maps without a common reference frame into one large global map. In their paper, they address map merging using motion planning algorithms. They perform operations such as translation and rotation on partial maps to determine common reference points among these maps until similar regions overlap. We use similar operations on partial maps generated by the robots to localize their position relative to the blueprint or the floor plan of the map being explored. We hope to extend our algorithm to tackle scenarios where this blueprint is not a part of the input and improve upon the efficiency of this early work. A popular researched problem in this context is called SLAM (Simultaneous Localization And Mapping), where the robot is required to build a map and localize itself at the same time. Nagesh Adluru et al present a novel present a novel and simple solution to the problem of map merging by reducing it to the problem of SLAM of a single virtual robot [Adluru et al.2008]. The individual local maps and their shape information would constitute the sensor information for this virtual robot. This approach allowed them to adapt the framework of Rao-Blackwellized particle filtering used in SLAM of a single robot for the problem of map merging.

3 Assumptions

We had to simplify our algorithm by making some assumptions that decrease the generality of the approach due to time and funding constraints. The main assumptions are as follows:

1. We assumed that the blueprints or floor plans of the building or map that is being explored to be given to us. This assumption, while simplifying the

problem a great deal, is reasonable, as search and rescue teams, more often than not, are provided with the blueprints on the scene. They can quickly upload these plans on to their robots in order for them to have a better understanding of the extent of the map.
2. We have a master-slave architecture for the robots (with one master commander robot and multiple slave explorer robots). The master robot is assumed to be located at a remote location such as the command post through which all information will be routed. It is also assumed to be fail-proof and computationally more powerful. The slave robots are the team of robots that are deployed on the field faced with the task of coordinated exploration. They could be subject to failure and have their navigation algorithms preprogrammed into them.
3. We assumed that perfect noise-free communication takes place between the master and the slave robots. Slave-to-Slave communication is not allowed at the moment. We consider communication in the form of a shared resource accessible by all the robots.

4 A Distributed Approach

We propose a distributed approach to the coordinated exploration problem. Our approach constitutes a master-slave architecture which is a model of intermittent communication where one device (the master) has unidirectional control over one or more other devices (slaves). The slave robots, called the explorer robots, are capable of mapping the environment on their own by individually exploring areas of the map assigned to them and collecting information to be communicated back to the master robot. The partial map explored by each of these slave robots is overlaid with the corresponding information that has to be communicated back; they can share information and coordinate their exploration activities with only the master. The master is set up at the command center. It lies at the other end of all the information communicated by the slaves. It is responsible for processing these different partial maps received from multiple slaves, localizing their position in the map and ensuring that all regions of the map have been explored for vital information while minimizing redundancies in the information being communicated and increasing the speed with which it is done.

5 Simulated Environment

We experimented our approach on a simulated 2-D grid environment. We consider an n X m grid where each cell block represents a unit of area explored by the slave robot. An example of a 10 X 10 grid is shown in Figure 1. Landmarks or walls are represented by the shaded region and they are assumed to be inaccessible. In other words, if movement is attempted into one of these regions, the slave robots will end up back in the same space. Empty spaces are accessible and are represented by an empty region. The slave robots flag these empty regions with their signature as they explore/visit them and move on. For testing

purposes, we assume there are 2 slave robots: A and B, whose signatures are A and B respectively. This algorithm can be extended to a grid of any size and any number of slave robots. The signature *-1* is assigned to *don't cares*.

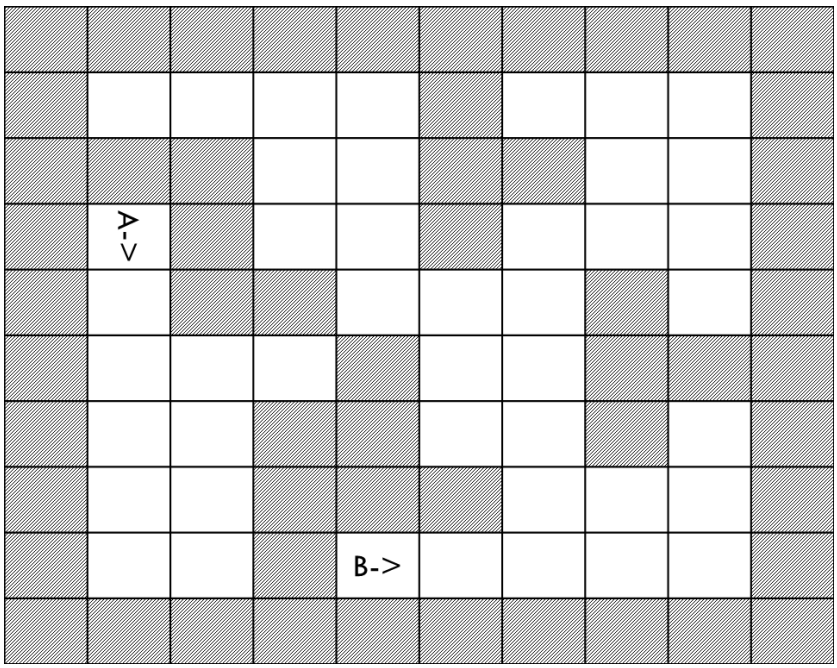

Fig. 1. An example 10 X 10 grid environment showing walls and accessible empty regions including randomly initialized start positions of slaves A and B and their respective orientations

6 Approach

Our master-slave architecture is designed as a multi-threaded client server system where a new thread is created for each slave robot which act as clients and the master is the server. Communication or information sharing happens only between the server and clients and not among clients themselves. The master (server) keeps listening on dedicated ports for each slave (client) and when the client is ready to transmit, the server is ready to receive the information and executes the task assignment algorithm. There are basically three steps involved: the pre-localization step, localization step and the post-localization step. Before we get into the pre-localization step, we will briefly describe the slave robot's navigation algorithm.

The slave robots are initially positioned at any random location in the map. So, as far as the master is concerned, their positions are initially unknown until

they have been localized. The slave robots maintain their partial map as they explore it and once a stopping criteria is reached, they share their respective partial maps with the master which then attempts to localize each of their positions. In this section we will discuss how a slave negotiates its way through the map until its time for it to communicate its findings and position relative to its starting point to the master.

The slave robots are allowed to move in four possible directions - North(N), South(S), East(E) and West(W). The robot keeps track of the path traversed by storing the coordinates of the cell blocks it has visited relative to its initial position and orientation. For the example shown in Figure 1, the slave A assumes its initial position to be (0,0). By moving to its north, it would have moved one block below where it is currently positioned in the blueprint. Each slave robot is preprogrammed to implement the following navigation algorithm. It is to be noted that this algorithm is not necessarily the most efficient.

6.1 Navigation Algorithm

The explorer robots function in two modes - the wall detection mode and movement mode. When the robot is in the wall detection mode, from its current location, it looks around and senses the presence of walls around it and flags each of them as a *wall*. It then computes the possible moves left based on this finding. First, the robot is initialized with equal probabilities to move in any direction. Once the available moves are computed, these directional probabilities are adjusted to only include possible moves. For example, in Figure 1, robot A detects walls to its south, west, and east. So it assigns a probability of one to go north as all other moves are not possible. When the robot is in the movement mode, it moves in one the directions using the directional probabilities as the likelihood. The robot keeps track of the relative coordinates of the grid traversed as well as the ones not traversed as it moves. The robot also maintains the count of the walls detected and the steps taken and once this count reaches a specified threshold, it communicates all the available information gathered to the master robot.

If the robot, in its current path, is at a position where there are no further moves possible because the cell block has already been explored or is a wall, it backtracks to the most recent untraversed block it had previously missed. The trajectory of the robot can be thought of as tree where the cell blocks are nodes and the possible moves are edges. For the example in Figure 1, robot A's trajectory tree including the directional probabilities would look like the one shown in Figure 2. The robot at (-2,2) has no where to go except backtrack to the most recent untraveled path shown in green to (-1,3). This minimizes overlap or redundancies in the exploration by exploiting its own trajectory first(using backtracking). In the case where all cell blocks in its own trajectory have been explored, the algorithm relaxes the overlap criteria and allow it move into the explored territory of the other robot.

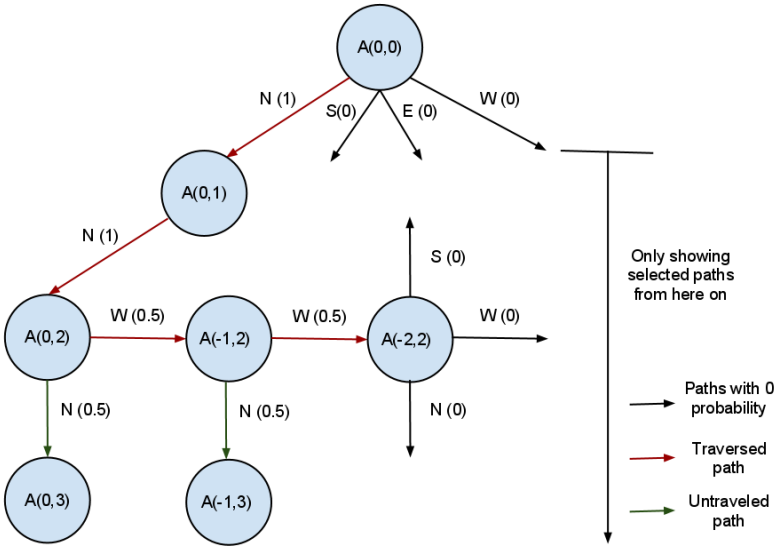

Fig. 2. Trajectory tree of robot A showing directional probailities

6.2 Pre-Localization

In this step, the partial map explored by the slave robot is communicated to the master. The master pre-processes this partial map for all possible initial orientations of the robot by rotating it by 90, 180, and 270 degrees. it then tries to superimpose each of these four versions of the partial map on top of the blueprint to check for matches. It is possible that this process could result in multiple matches. This implies that the master is still uncertain as to what the inital position of the robot could be. The Figure 3 shows a possible scenario.

6.3 Localization

In this step, since the master is still uncertain about the absolute position of its slaves, it instructs them to continue their navigation but this time communicate the partial map back each step of the way until the master is able to clear the ambiguity. This may be extremely inefficient in maps that are sparse but very effective in relatively populated maps. Let us show how this works using the running example in Figure 3. As it can be noticed, there are two possible matches returned. So the master instructs the slave to execute one more step (say the robot ends up moving west) and its relative position is now (-2,2). The new partial map is now communicated back to the master which in turn finds only one suitable match and is shown in Figure 4. This allows the master to trace the steps back to the robots initial position and thereby localizing its accurate absolute position in the blueprint. Now the master is in a position to assign more informed tasks about what the slaves should do next.

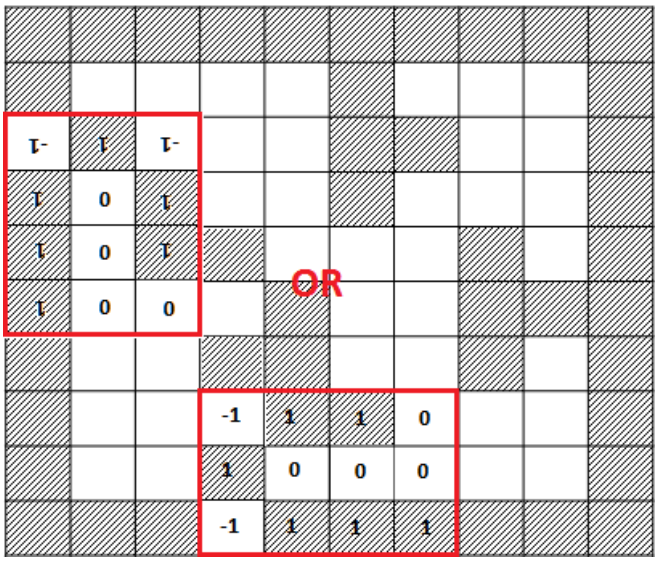

Fig. 3. Pre-Localization Step: Two possible matches found when the different versions of partial map is superimposed over the blueprint

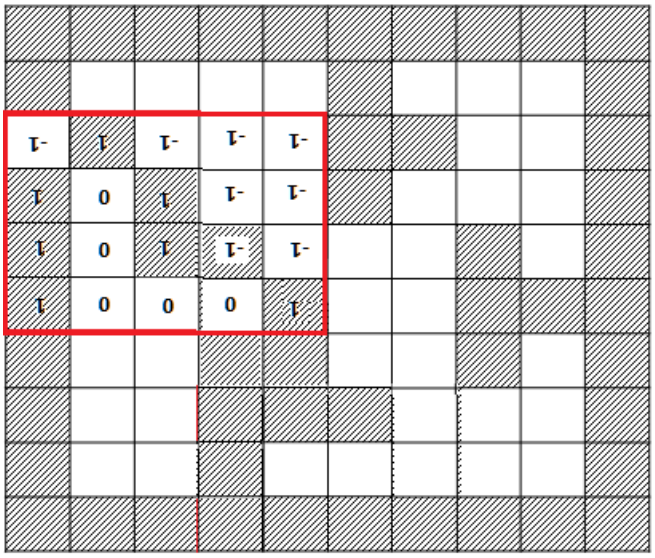

Fig. 4. Localization Step: Only one possible match found when the new partial map after one more step (going west) is superimposed over the blueprint

6.4 Post-Localization

The master now has accurately computed the absolute position of the slave robot and is ready to assign its next task. In this step, the master instructs the slave robots to continue exploration as per their navigation algorithm except with a newly computed set of direction probabilities. This computation weighs in on the location of unexplored areas, current absolute location of the slaves and the trajectory so far. Higher weightage is given to the directions that lead the slave to unexplored regions. Penalties are awarded for overlaps thereby decreasing the probability assignment to already explored regions by other robots.

7 Future Work

Currently, in the event of a faulty slave failing to communicate, since the intermittently updated blueprint is the only thing shared, the master will continue to not reflect areas explored by the faulty slave in the blueprint, thereby forcing the remaining slaves to reexplore them. As per our assumption, the master is fail-proof. To make the situation more realistic, we could allow robots to 'die' due to a sensor malfunction or a dying battery or communication failure causing robots to lose contact with one another or a structural collapse on the robot. Thus, there has to be a fail-safe backup algorithm for reassigning roles in the event of robots (master/slave) dying and dealing with information loss. As you may have noticed, our approach is centralized and depends on the master for communication and future task assignment. This is a single point of failure and if the master crashes the whole operation crashes. However, having a pre-assigned remote master helped reduce the complexity of the problem by not having to consider an election algorithm to elect a new master in the event of a master dying or a fail-safe algorithm to handle information loss due to such an event. Also, having a remote master meant that the computational capability of the slaves could be restricted to fulfilling their specific task alone and the master could be the only superior (and in turn, more expensive) system in terms of computational power in order to carry out the map exploration algorithm. However, this could definitely be an extension of this work.

We could also investigate scenarios where the blueprints of the map is not given to us. This would indeed make the problem much more complex because the partial maps will not have a common referrence point by default and that needs to be computed. There is also the possibility of certain partial maps not having any overlap in which case the master has to maintain multiple maps and continuously keep checking for the possibility of overlap.

This project lacks in experimentation and testing due to lack of time. Assuming there is a cost for communication, we would like to study the trade-off between the efficiency of the map exploration algorithm and the speed with which it is carried out. We would like to analyze this by varying the number of steps 'k' taken before communication and study the trade-off parameter. Intuitively, higher the value of 'k', the larger the area covered within one exploration and feedback step, the greater the possibility of overlap in the areas explored by

different robots, the lesser the efficiency of the algorithm. Alternatively, lower the value of 'k', the lesser the area covered before the robot reports the partial map to the master, lesser the chance of overlap in the area covered between the robots, but larger the cost incurred for communication. It would also be interesting to study scenarios where the communication is noisy.

We would also like to optimize the number of robots needed to efficiently carryout the map-exploration algorithm assuming there is a cost for including each robot in the team and also compare the performance of our algorithm with existing techniques in terms of the costs, time taken, accuracy of the reconstructed map, and the efficiency of the algorithm.

We would also like to implement this on real miniature robots in real environments.

8 Conclusion

Project Maera is a distributed robotic map exploration project which uses image processing techniques such as rotation and translation to localize robots in the blue prints provided. We presented a simple yet effective localization approach. We implemented the algorithm using a multi-threaded client server architecture and verified that the algorithm works. We attempt to minimizes map overlap by allowing the slave robots to explore its own trajectory first using backtracking and if needed move into the territory of other robot. Due to time and money constraints, we have a huge list of things to improve our approach that are still pending. However, this project could be a good starting point for bigger and better things to come.

References

[Adluru et al.2008] Adluru, N.; Latecki, L.; Sobel, M.; and Lakaemper, R. 2008. Merging maps of multiple robots. In *19th International Conference on Pattern Recognition*, 1–4.

[Carpin, Birk, and Jucikas2005] Carpin, S.; Birk, A.; and Jucikas, V. 2005. On map merging. In *International Journal of Robotics and Autonomous Systems*, 1–14.

[Thrun, Burgard, and Fox2000] Thrun, S.; Burgard, W.; and Fox, D. 2000. A real-time algorithm for mobile robot mapping with applications to multi-robot and 3D mapping. In *ICRA*.

[Yamauchi1998] Yamauchi, B. 1998. Frontier-based exploration using multiple robots. In *Proceedings of the Second International Conference on Autonomous Agents*.