

A Communication Model for Completing a Task using Multi-Robot Coordination

Cheol-Woo Jung, Pushkar Kolhe, C Song

April 17, 2008

Abstract

Single robots are insufficient to perform multiple or complex tasks such as exploration or de-mining, multiple robots can be used to increase efficiency and reliability. Here coordination and planning among the robots is important. The project focuses on such an issue. There are many researchers who have proposed different models for multi-robot coordination. We have tried to implement such a model so that we can realize the power of the multi-robot system along with the difficulties involved in a multi-robot system.

1. Introduction

Multi-robot systems are more powerful than single robot systems because they can complete tasks, in more than one instance, faster and efficiently. Understanding multi-robot systems is also essential from the viewpoint of failure in a robot system. That is precisely what we address in our project. It is quite possible that in the future when 10 robots are designed for a specific task, for example in a war, we may send 20 or even more just in case some of them fail. However, some tasks can be critical in a sense that if there exists a task that no one robot can complete, it is important that instead of retiring the multi-robot system should find a way to complete the task by helping each other out.

2. Previous Work

There has been much work done in Multi-robot Coordination over the past few years. (Rafael Fierro, 2002) describes previous work in this area in detail. Most of the work has involved experiments like search and rescue, exploration, surveillance and distributed manipulation. Most of the work essentially deals with how a robot can access a service of other robot, for example, by using a hierarchical architecture as in (Rafael Fierro, 2002). One of the other approaches involves using an auction-based coordination system where each robot can auction a service that needs to be performed over a sub-system. The bidder who wins this auction will perform the task. (Mataric', 2000) argued that this model is flexible in terms of number of robots and robots can be dynamically added. This proves that this is a rather new area for research and many new interesting developments are shaping up. (Stroupe, 2001) has interesting an interesting idea where sensors in multi-robot systems were used to make a distributed sensor model that was used for localization. This was very interesting. This prompted us to think about multi-robot coordination on a lower level than behaviors, that is, the sensors.

3. Experimental Setup

Our task here is exploration. One of the robots has explored its environment, while the other cannot do so because of its limited sensors. We assume that the robot with limited sensors has lost some of its sensors or they are corrupted. So, it loses some of its behaviors and now it has to rely on other robots to

perform its task. We ran simulations of a 2 robot system in Microsoft Robotics Studio. A motivating scenario for this idea is surveillance. The robots we used were:

1. A Pioneer P3DX Robot which is equipped with a SICK Laser Range Finder and shaft encoders.
2. Irobot CREATE Robot which is equipped with shaft encoders and a webcam.

In our experiment the Pioneer robot used EKF Slam to find a global map and localize itself. The Create on the other hand cannot really create a global map or go to any location using the shortest path. But, it can take pictures from its webcam. Such a robot can be used for surveillance. The entire surveillance procedure takes place in this way. Firstly, the Pioneer robot creates a line-based map. Then the Pioneer and Create robot communicate and determine which sensors each of the robots has and hence which behaviors they can execute. When a certain goal location is given to them, they use these known behaviors of both the robots to complete the task in the most efficient way. Efficiency in this experiment is just saying that the task was completed without crashing into any obstacles by any robots.

The Navigation and Mapping used by Pioneer is discussed in Section 1. The Communication model is discussed in Section 2 and the Planning is discussed in Section 3. The discussion for this project is limited to the sensors that are present on the 2 robots. But, in general we think that this model can be suitably generalized.

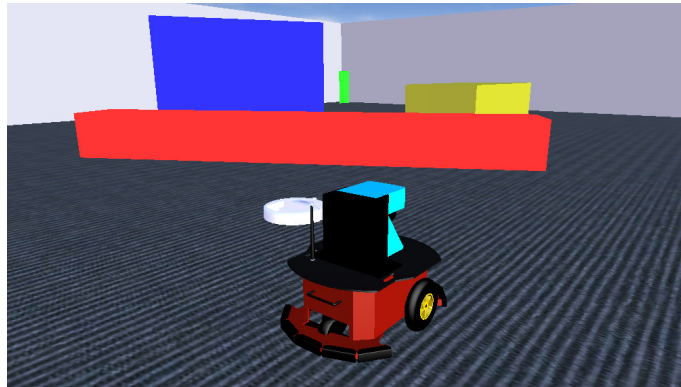


Figure 1 Simulation Environment. The target here is to Grab Frame at the small Green Pole at the center of the Image

4. Navigation and Mapping

We have used a line-based mapping technique for SLAM. The line based technique has advantages because unlike grid based maps it is not restricted by resolution. It is very easy to determine how far the walls are from any position on the field and this can be easily interpreted by the Create robot with the bumper sensor.

A thorough summary of the implementation of the EKF SLAM is in Section

5. Communication and Planning

Communication in Multi-robot systems is very essential. Planning after getting information from communication is even more important. In the first stage of the communication process, the robots communicate to each other the sensors they possess. This helps each of the robot figure out what behaviors the other robot can have. A particular task is then given to one of the robots, the Create in our case. For the experiment we conducted, we asked Create to retrieve an image of a particular object present in our field. On listening to this request, the Create also gets a list of behaviors that it needs to complete this task. The Create robot uses its previous knowledge and determines that it can ask the Pioneer robot to help him with certain behaviors. When the Pioneer responds, the Create creates a plan of execution of behaviors that both of the robots need to execute to complete this task.

We predict that such a mechanism can be extremely effective. If a robot understood what sort of behavior each sensor can give it, theoretically, it should be able to use this information to realize its shortcomings and only perform tasks that it knows can be best performed by it.

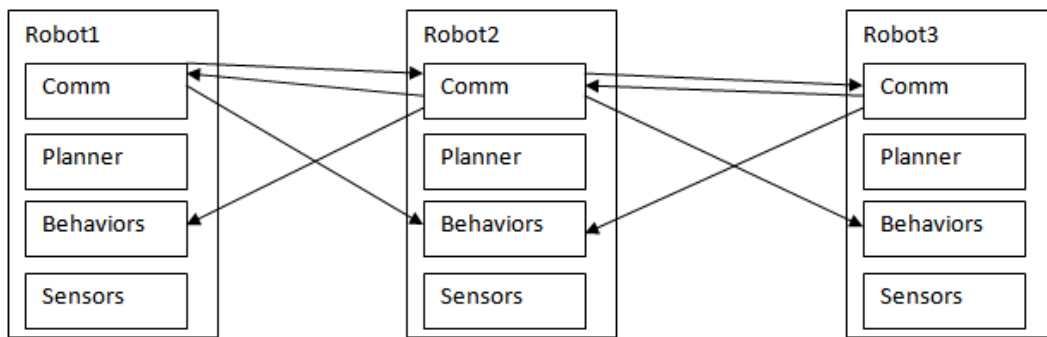


Figure 2 Communication Model in Robots and Distributed Behaviors

In this paper we focus on a model that can accommodate failure in sensors of some robots. This is simply done by recognizing the sensors on each robot with their behaviors. Hence, the communication modules in Figure 1 are linked to the Behaviors Module. Hence, any robot can give commands to activate a certain behavior of the robot. Planner is not involved here. The Planner only works when a task is given to that robot. The Planner then will create a list of behaviors of other robots, arrange them in an order of skills so that it can create a plan.

Every robot has a set of sensors and a hence a set of behaviors that it can apply itself to. Every robot makes each of its behaviors available to other robots in the system. This is done by a simple query-message model, where to every query, the robot replies with a message containing the sensor hardware it has. Every robot also has a planner which takes in the task given to the robots by some user. The Planner knows the number of robots in the system and behaviors for each robot. It stores all this information in a list of behaviors where the robot id is either present for that behavior or not. Every behavior is associated with a skill level. The skill level is sort of a tie-breaker when two or more robots can do the same behavior. The skill of a robot for using a particular behavior increases when it uses the

behavior to complete a task successfully. The planner uses this list of behaviors while trying to plan for a particular task.

For the purpose of this project, we have assumed the planner is very simple and it has templates for each and every task that it can ever get. So, for the simple task of the Create to go to a position B on the field and grab an image, it knows from a script that it needs to have behaviors like 'Mapping, Path Planning, Navigation, Obstacle Avoidance, Frame Grabbing.' It chooses itself for every behavior it can perform like Navigation and Frame Grabbing. For the others it searches for a robot with the highest skill for that particular behavior, here the Pioneer robot. Though this model seems a bit restrictive, it is easy to add test cases.

6. Execution of Tasks

Once the plan was decided, the execution of the tasks had to be done. For this, the robot that created the list of behaviors to be implemented sends out method calls, which are actually, some behaviors that the robot can perform like Navigation with appropriate parameters, like the end point for Navigation. The robot requests appropriate messages from the Navigator robot so that it can use it for Navigating along with it. For Navigation, they are a series of waypoints that leads to the final destination. The waypoints eventually given by the Navigator robot will be such that they will not crash into any obstacle as it has an Obstacle Avoidance behavior in it.

7. Implementation

7.1. EKF SLAM

The program was written in C# and C++ using Microsoft Visual Studio and Microsoft Robotics Studio Simulation Environment.

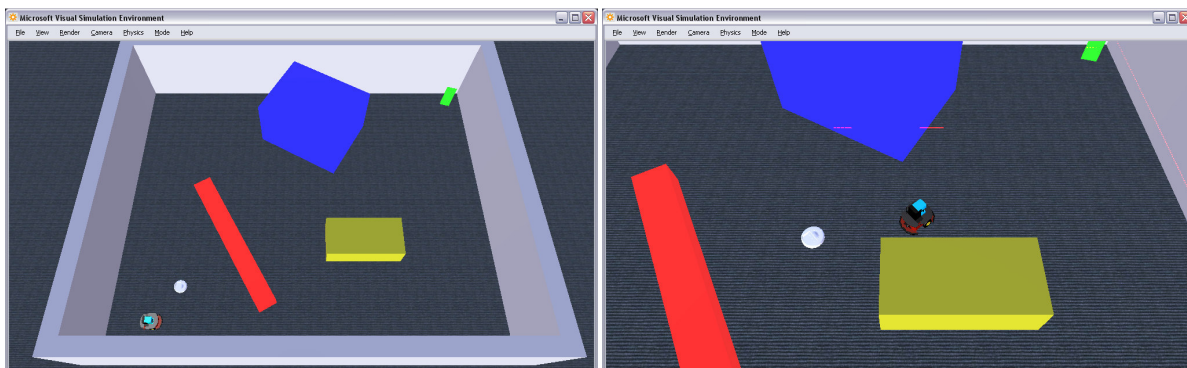


Figure 3 Left: The simulation environment which consists of Pioneer, iRobot Create, obstacles and walls; Right: Simultaneous Localization and Mapping (SLAM)

The SLAM deals with the problem of building a map of an unknown environment incrementally while navigating the environment using the map. This problem is important because it makes the mobile robot truly autonomous. On the other hand, it is also a hard problem because of lots of uncertainty such as sensor noise. For our project we implemented the Extended Kalman Filter based SLAM algorithm using

line features extracted from the SICK laser range finder. It was possible to use a priori map of the environment, but for a reasonable complexity of the project we decided to create a map using SLAM.

7.1.1. Map Representation

There are several ways to represent the world in the map such as grid, topology, and features. Among those we chose features and especially line segments. Line segments are very common and basic geometric features which can be observed easily from the environment. It might not be a good choice if it is the outdoor environment, but since we assumed that our simulation and test environments are indoor, it is easy to extract the line segments from the surrounding environment.

7.1.2. Line Extraction

Viet et al. evaluated and compared several different well-known line extraction algorithms on 2D laser scanner, which are split-and-merge, line regression, incremental, RANSAC, Hough transform, and EM. Depending on the evaluation criteria each algorithm has its own advantages and disadvantages, but split-and-merge and incremental algorithms shows better overall performances over others. Since our assumption lies on the simple environment, we have chosen split-and-merge algorithm for extracting line features from laser scan data. While it is simple and easy to implement, it also shows good results.

7.1.3. EKF (Extended Kalman Filter)

In order to estimate the pose of the robot the extended Kalman filter was used. The EKF is basic and yet widely used localization algorithm. Since the Kalman filter is an optimal state estimator, it converges at least for the linear system and the EKF is adapted for nonlinear system. Even though the EKF is efficient, it also has a few drawbacks. First of all, it is based on uni-modal Gaussian distribution model, which is not the case for the common nature. That is, if the robot predicts the wrong state, there is no way to go back and correct it. Moreover, since the EKF keeps track of all the states of features as well as robot position, it could be computationally very expensive with large number of landmarks. However, our simple environment assumption assured that the EKF is effective. The implementation based on the EKF is as follows.

1. Time update: Estimate the current pose of the robot using the odometry data (control model)
2. Measurement update: Upon arrival of new measurements, update the estimated state based on observation model
3. Add the new landmarks to the state and also update uncertainty

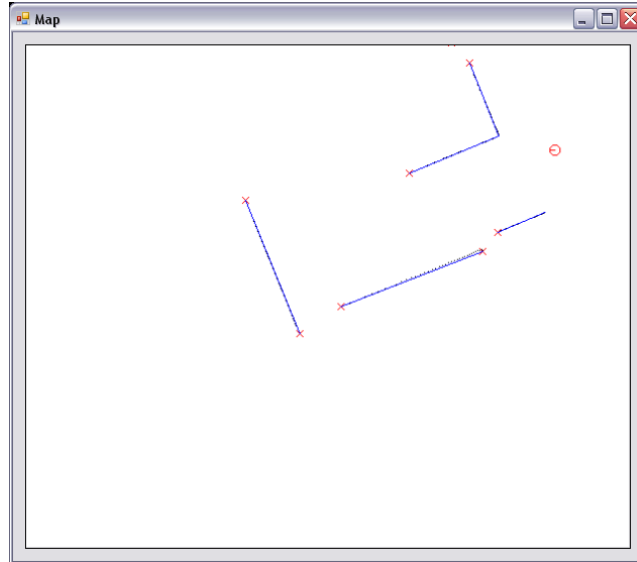


Figure 4 Building a map using SLAM. The blue lines are extracted features, the read cricle is the robot, and the red cross is end points of the line.

7.2. Communication and Planning

Let us consider the example of our experiment where the Create robot has to go to some goal position (Pos A) and take an image. Every robot has a pre-known table of entities which relate sensors to behaviors. For example,

```
Sick LRF { Obstacle Avoidance, Path Planning}
Camera { Record Images, Segmentation, Tracking}
Encoders { Localization}
Bumpers { Obstacle Detection}
```

Table 1 Sensors and Behaviors

When a robot queries another robot on a network it uses the WinSock library of Microsoft to listen to messages. Every robot responds with its id, sensors and skill sets for each behavior of a sensor. This can be one or two of the statements of Table 1.

```
Obstacle Avoidance { Pioneer (90)}
Grab Frame { Create(100)}
Navigation { Pioneer (70), Create (20)}
Bumpers { Create (100)}
```

Table 2 List of Behaviors with each Robot

```
Task-Capture Frame at Pos A
  Navigation to Pos A
  Obstacle Avoidance A
  Grab Frame at A
```

Task-Something else
Behavior1 Parameter1
...

Table 3 Script for a Task

The robot who is assigned a task, then uses this information and arranges them in a list of Behaviors, including its own behaviors. The numbers in the bracket indicate skill level of that robot. Then when a robot gets a particular task, it uses the script provided to figure out the behaviors it requires. For example, for Frame Grabbing, it realizes that it is supposed to perform Navigation and then Grab Frame. In our implementation, we have simply used robots with higher skill level to implement a behavior. So, the Pioneer would do the Navigation to Position A while doing Obstacle Avoidance. The path that the Pioneer gets will be passed as a message to the Create. The Navigation and Obstacle Avoidance always returns a path as a return message. Grab Frame can only be performed by the Create and it then does that. So, in this way, the Create can achieve the task of grabbing a frame at some position on the field.

7.3. Path Planning

We have used A* Algorithm in our simulation for the Navigation Behavior of the robot. The program was implemented in C++. Please refer to the Appendix A to get details on it.

8. Results

8.1. Limitations of the System

We implemented the following code on two robots. After adding more scripts in our planners, we think we can implement more tasks. The communication model that we are using currently is a TCP/IP oriented protocol. Hence, each robot should know what other robots are present in the world. This was just done for the purposes of faster development. If we can use a form of multicast or the UDP protocol we can scale this model and even add a 'discovery' behavior where each robot can find new robots. Along with these shortcomings, the model may suffer bandwidth problem when introduced in a swarm of more robots as messages exchanged between them hugely increases. Some of the advantages of this approach are that one robot gets to decide a plan to achieve a task. This means that there is no competition or communication necessary for robots to agree on a task. This saves time and reduces complexity. Also, since this is a heterogeneous system, it makes sense for a robot to take its own decisions.

8.2. Were the goals met?

We developed a model where the two robots communicated and called each other's behaviors. We worked this out in the Microsoft Robotics Simulation. We later realized that the Pioneer robots in the lab do not run MSRS and the EKF Slam code was a bit too complicated to rewrite in Linux or Mission Lab. Hence, we considered that the mapping problem can be solved by using EKF Slam Algorithm as showed above. Later, we wrote some code in the ARIA interface for Pioneer for exhibiting behaviors as navigation and obstacle avoidance. We were also able to achieve successful communication of message.

We faced some problems with sensors on the Pioneer and hence, we will be showing a live demo after this paper submission is done. Some simulation pictures are shown in the paper and a demo will be shown in the presentation.

Overall, we determined that our method works with two robots in the simulation.

9. Work Done by Team Members

Choel-Woo Jung

CJ developed the EKF Algorithm in Microsoft Robotics Studio and worked on the simulation. Since, the simulation works on a single PC, the simulation does not actually communicate in the networking sense, but it uses similar data structures.

Pushkar Kolhe

Pushkar developed the behavioral code for the Pioneer robots such as Obstacle Avoidance and Navigation using the ARIA interface in Visual C++. He also developed the Communication Model.

C Song

C Song developed the Path Planning A* algorithm which was used as a message that was passed from one robot (Pioneer) to Create for Navigation.

10. Works Cited

Fox, S. T. (2005). *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press.

Mataric', B. G. (2000). Agents, MURDOCH: Publish/Subscribe Task Allocation for Heterogeneous. *In Proceedings of Autonomous Agents 2000*, (pp. 203-204). Barcelona, Spain.

Nguyen V. Martinelli, A. N. (2005). A Comparison of Line Extraction Algorithms Using 2D Laser Rangefinder for Indoor Mobile Robotics. *Intelligent Robots and Systems, 2005. (IROS 2005). 2005 IEEE/RSJ International Conference*.

O., A. K. (2003). *Feature-Based Mobile Robot Navigation in Known and Unknown Environments*.

Rafael Fierro, A. D. (2002). A Framework and Architecture for Multi-Robot Coordination. *The International Journal of Robotics Research*, (pp. 977-995).

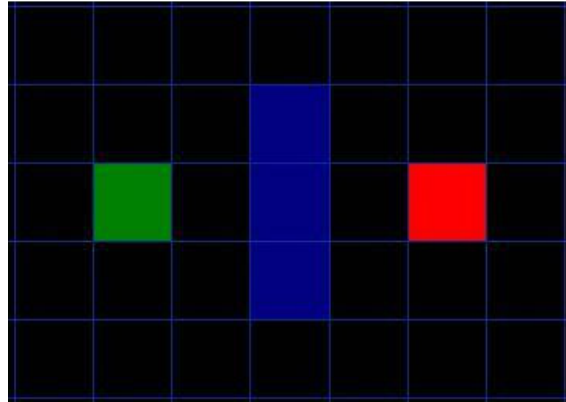
Stroupe, A. M. (2001). Distributed sensorfusion for object position estimation by multi-robot systems. *In Proc. IEEE Int. Conf. on Robotics and Automation*.

Tucker Balch, P. *Robot Team*.

Appendix A

A* Algorithm

Assume that we want to go from point A to point B. Let's assume that a wall separates the two points.



We begin the A* algorithm by doing the following:

1. Add the starting square (or node) to the open list.
2. Repeat the following:
 - a. Look for the lowest F cost square on the open list. We refer to this as the current square, $F = H + G$, where, H = the movement cost to move from the starting point A to a given square on the grid, following the path generated to get there and G = the estimated movement cost to move from that given square on the grid to the final destination, point B.
 - b. Switch it to the closed list.
3. For each of the 8 squares adjacent to this current square ...
4. If it is not walkable or if it is on the closed list, ignore it. Otherwise do the following.
5. If it isn't on the open list, add it to the open list. Make the current square the parent of this square. Record the F, G, and H costs of the square.
6. If it is on the open list already, check to see if this path to that square is better, using H cost as the measure. A lower H cost means that this is a better path. If so, change the parent of the square to the current square, and recalculate the G and F scores of the square. Resort the open list.
7. Stop when:
8. Add the target square to the closed list, in which case the path has been found (see note below), or Fail to find the target square, and the open list is empty. In this case, there is no path.
9. Save the path. Working backwards from the target square, go from each square to its parent square until reaching the starting square.