RoboCup Rescue 2014 – Infrastructure League
Team Description
S.O.S (Iran)

S.Mohammad Reza Modaresi, Yoosef Golshahi,
Fatemeh Pahlevan aghababa, Pegah Taheri, NavidHaeri, Salim Malakouti,
Angeh Aslanian, Aramik Markari, Morteza Rezayi[1]

[1] Robotics Research Center, Department of
Computer Engineering and Information Technology,
Amirkabir University of Technology,
No. 424, Hafez Ave.,
Tehran, Iran
modaresi@aut.ac.ir, {yoosef.golshahi, reyhaneh.pahlevan,
navid.it.haeri}@gmail.com, salimm@cs.pitt.edu,
{ angeh.a2, aramikm, m.rezayi69}@gmail.com
http://www.sosrcrs.com/

Abstract. One of the most important goals of Robocop Rescue simulation is providing a base to help researchers implement their strategies. So we suggested adding Agent Development Kit (ADK) as Infrastructure field competition. S.O.S team has a very pleasant background during 12 years of RoboCup rescue simulation agent. Our team has achieved more than 8 trophies during these years. S.O.S has designed a powerful base during these years and now we are focusing only on our strategies. Our multi-layer and state-base code design, helps us separate agents' strategies. Because of our low level strategies such as clearing a road, extinguishing a building or rescuing a civilian, agents concern about choosing the best target instead of dealing with how to perform these tasks. This base allows us to use, implement and test many high level strategies and AI methods.

1 Introduction

We designed a multilayer and state-base code design because it helps us to separate concerns. It is a powerful design for huge codes. We will explain it in details in following sections.

2 Software Architecture

S.O.S agents 2014 are based on SOS 2009 that described in the paper we published 4 years ago. In 2010 we changed it duo to kernel changes and tried to integrate it into

a newer version. We tried to find base problems and solve them. Now we have developed a collection of very good tools like reachability, message system, sensible area, fire sensible area and etc. These are the tools used for developing rescue agent strategies. The agent strategies' structure is defined as follow:

2.1 Agent design and code structure

We designed a multi-layered structure, because we believe it is easier to optimize and debug such structures. We also could divide the decision-making process of the agent to different –higher and lower- levels. Therefore as can be seen in the figure we have designed four layers.

2.1.1 High level decisions

This level chooses which state should be taken care of at the moment. It checks the state that the map currently has. It changes the priority of tasks considering the situation of the environment such as blackness and size of fire zones. This is the only part that we have been trying to train. It will let agents be flexible in different maps and scenarios.

2.1.2 States decision

For each situation we should have a plan. States are the activity of a situation that we have planned. This makes high-level decisions to decide easily and makes it easier to handle the situation without considering other situations.

2.1.3 Low level

In this level we use methods implemented in S.O.S basic agent and low level acts such as clearing a blockade. We make the decision that, in order to clear a road, which blockade should be cleared first and how should it be cleared.

3 Agent skills and action selection

Most of agent's general skills (i.e. low level abilities) are as it was in previous years. For example, in the path planning strategy, we use a special version of Dijkstra single source shortest path algorithm [4], using a priority queue implemented by S.O.S. team whose time complexity is e log(e), where e is the number of edges in the city graph (i.e. roads) and due to the fact that we consider the maximal sequence of

roads between two junctions which has no junction inside as a single road, the complexity decreases significantly.

Action selection of every agent is through a special architecture, which is described in the software architecture section.

3.1 Reachability recognition method

We believe in the new server an important and major problem is to find point-to-point and area-to-area reachability. In the previous server, roads were edges of the city graph so the problem was bounded to find area-to-area reachability, but in the new server roads and buildings are areas with two dimensions therefor in order to specify if two areas are reachable to each other we should first be able to find point-to-point reachability. And to make a world graph we consider that each passable edge of road is a node and the edge is a direct line between these nodes in a road.

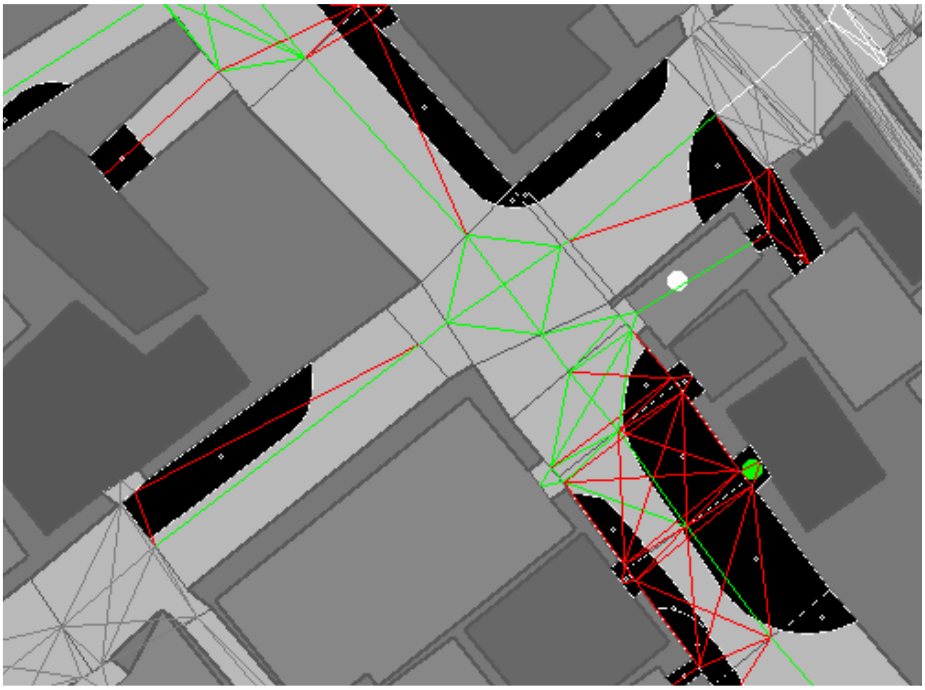


Figure 1: Green Edges are open, Red one is close and gray is foggy close

3.1.1 Point-to-Point Reachability

To achieve this end, we expand all blockades and roads edges 500mm (1/2 of agent's dimension)then we intersect the new shapes together and we find reachable parts. These parts in Figure 2.C are shown by numbers and are surrounded by blue lines. An agent can fit in the road if its center is in any of these reachable parts or areas. Therefor in order to find point-to-point reachability we have to check if the two points are in the same reachable part.

Figure 3.A

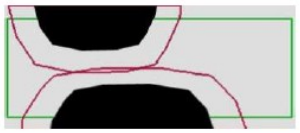


Figure 2.B

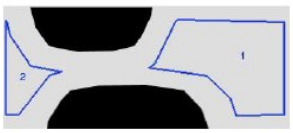


Figure 3.C

3.1.2 Area-to-Area reachability

Now that we have point-to-point reachability we can find global reachability too. We find our global reachability, using path planning algorithms and methods to keep reachable parts – similar to keeping fire zones – in order to reduce the time complexity. We managed to design a reachability structure in our basic agent that represents agent's reachability to any area and point of the map with a low time complexity of O(1) . We also manage to find reachability of two areas of the map which none is the position of the agent using reachable parts with a low time complexity.

4 Agent coordination and communication

Depending on the strategy each agent decides in a specific situation, the decision will specify whether to work centralized or distributed. However, center agents think that their platoon agents are working centralized so they provide centralized information needed by platoon agents. As the platoons have almost the same world model, their decision about this matter will be coordinated sufficiently.

4.1 Message System

A new implementation of channel-based communications has been used. This Model limits voice channels by range, message size and number of messages, and limits radio channels as following:

Messages are heard at the start of each time step as part of the sense cycle and both voice and radio channels may have random noise added.

The more information about our flexible Message System is in the Infrastructure paper.

4.1.1 Possible scenarios:

Possible scenarios could include:

1. One or two low-bandwidth, high reliability channels and several high-bandwidth, low reliability channels.
2. A large number (10 - 20) of low-bandwidth channels.
3. One high bandwidth, high reliability channel and a number of high-bandwidth, low reliability channels.
4. Only one channel with moderate bandwidth.
5. Only one low-bandwidth channel.
6. No radio channels at all.

4.1.2 Center and MiddleMan

In this communication model there is no difference between Centers. So we make a virtual center for each agent that is named "Center Activity" for doing activities of real centers so we don't use real center if they aren't useful.

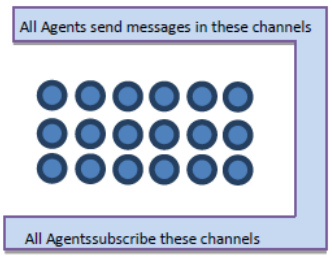
We defined "MiddleMan" as an agent (center or human) that is placed between the paths of sending and receiving message in order to collect messages and summarize messages to send to other Agents.

4.2 Channel Distribution

According to possible scenarios channel distribution can be divided into three states.

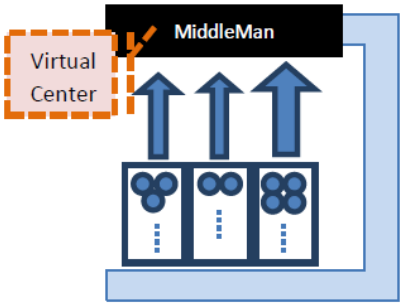
4.2.1 No MiddleMan

At first it is better to have "No MiddleMan" communication, because the delay of each message is just one cycle. In this state, the most bandwidth channel is used and agents are assigned to these channels and get almost the same message size limitation. All agents subscribe these channels therefore each agent receives both its message and other agents' messages.



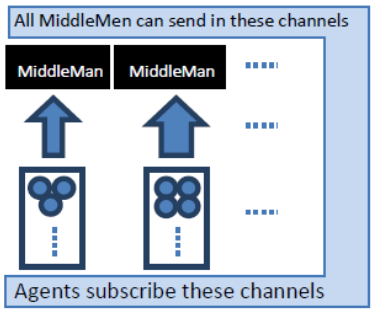
4.2.2 Central MiddleMan

The MiddleMan chooses the channels with highest bandwidth from remained channels (according to subscribe limitation for the MiddleMan) and all agents (except the MiddleMan) send their messages to these channels. In this state the MiddleMan do thinks of all center activities and if it is a human, the MiddleMan also does his activities.



4.2.3 More than one Middle Man

If we should have more than one Middle-Man, each message has 2 cycles delay. The worst thing in this model is that it takes 4 cycles for agent between the times it sense the changes in current position and receive a message from center activity about this situation. This state is like "Central Middle Man" but MiddleMen don't do activities of center activities because these MiddleMen don't have complete data of world model.



4.3 Noise

Messages may be dropped by "Noise". We don't have any data about noise probability. So we can't involve noise in channel distributing. All we can do to decrease effects of noise is detecting messages that have not been received and sending them again. For instance If messages size is less than bandwidth some important message will duplicate so it is less likely for important messages to be dropped by noise.

4.3.1 Detect Noisy Message

After studying many technics such as acknowledgement system and… we found out that other header that added to each packages (ex. for acknowledge) reduce performance so we decide to use a channel distribution model that every agent can hear their own message so it can detect messages that are not received by itself. If an agent doesn't receive his message till ex. 2 cycles (it depends on priority of message block) the agent adds the message to noisy message buffer.

4.4 High level Strategy

In order to choose channel distribution, we check if without a MiddleMan each agent can send 30bytes, or if the number of all channels is less than agents subscribe channel or if MiddleMan is just a broker, we use "No MiddleMan" Communication.

Otherwise we need at least one MiddleMan. At first we try to have just one Middle Man but if we can't have a good communication with one MiddleMan we add other MiddleMen till there is no remained channel or we have a good communication or received message bytes from the MiddleMan are less than channels that could be used to send.

The possible scenarios may include:

1. One or two low-bandwidth, high reliability channels and several high-bandwidth with low reliability channels.

2. A large number (10 - 20) of low-bandwidth channels.
3. One high-bandwidth, high reliability channel and a number of high-bandwidth, low reliability channels.
4. Only one channel with moderate bandwidth.
5. Only one low-bandwidth channel.
6. No radio channels at all.

Thus different strategies should be used for too specific scenarios, such as scenarios with one low bandwidth channel, low reliability channels and no radio channels. Many of these strategies were previously explained, thus, we only described the following section:

4.4.1 Strategy for one low bandwidth channel

For low bandwidth channel, a new kind of message blocks are used and only one message block is sent per package. This package's size is only 16 bits. At first the fire message will be sent, and then only civilians' positions are reported. The message is sent as soon as the agent considers it important regarding its priority.

4.4.2 Strategy for low reliability channels

We believe that, receiving important messages in the low reliability channels is more efficient than getting too many messages with normal and important priority. Since the probability of having noise in the second packet it at most $p_f * p_f$, we decided to duplicate important messages. However, duplicating messages results in wasting bandwidth. Therefore, the messages are sent normally and the important messages are only duplicated using the aforementioned method in previous section.

5 Software Tools

We utilize eclipse as part of our IDE and Ubuntu as operating system because of its high performance and finally we use SVN for code version control. And also we provide some other tools for debugging our strategies and base.

The most important tool is Agent World Model Viewer, which provides an easy usage interface and adding layers are made so easy. Currently we have about 115 different layers that are responsible for different strategies. This is our tool for virtual debugging. The next tool is Agent Logger, which logs things that happen in the code. This helps us find the problem when we are playing the logs. Other important tool that we use is log viewer. We modify the log viewer to be able to rebuild agent's world model by parsing the communication and agent sense.

6 Acknowledgements

We have great thanks to our teacher Prof. Homaioonpour and Computer Engineering and Information Technology Department of Amirkabir University for Robotics Innovation Lab, which provide us a research environment and inspires our work.

Also, we specially appreciate Mobinnet Company for their efforts, sponsorship and support.

7 References

1. A General Computational Recognition Primed Decision Model with Multi-Agent Rescue Simulation Benchmark by Alireza Nowroozi; Mohammad Ebrahim Shiri, Assistant Professor; Angeh Aslanian; Caro Lucas, Full Professor
2. Golshahi, Y et.al: S.O.S. Team Description Paper Proceeding of Robocup 2013.
3. Modaresi, M et.al: S.O.S. Team Description Paper Proceeding of Robocup 2012.
4. Markari, A et.al: S.O.S. Team Description Paper Proceeding of Robocup 2010.
5. Hashemi, B et.al: S.O.S. Team Description Paper Proceeding of Robocup 2009.
6. Ghaffuri, M et.al: S.O.S. Team Description Paper Proceeding of Robocup 2008.
7. Azizpour, H. et.al: S.O.S. Team Description Paper Proceeding ofRobocup 2007.
8. Ansari, M. et.al: S.O.S. Team Description Paper Proceeding of Robocup2006.
9. Cormen, T., Leiserson, C., Rivest.: Introduction to Algorithms MIT Press, Cam bridge(2000)
10. Horstmann, C: Object-Oriented Design and Patterns
11. R.C. DubesandA.K.Jain. Algorithms for Clustering Data. Prentice Hall, 1988.
12. Robotic Rescue Simulation league Rules.