RoboCupRescue 2013 – Rescue Simulation League Team Description

<ZJUBase (P.R.China)>

Institute of Cyber-Systems and Control
Zhejiang University,P.R.China
liuxiaodoubao@126.com
http://www.nlict.zju.edu.cn/zjubase/index.html

Abstract. In this document, we describe some specific features of ZJUBase rescue simulation team applying for RoboCup 2013. Model building and algorithm implementation of the main aspects of Rescue Simulation are discussed. Flexible software architecture, multi-mode path planning, self-adaptive communication model, roads classification, partition algorithm, agents strategy and some other primary technologies are depicted in detail. As we have participated in RoboCup 2012, this document considers more of the major updates we have made from the TDP of 2012 [1].

1. Introduction

The RoboCup Rescue Simulation environment is of great social value and it provides a platform for the development of algorithm design, artificial intelligence, statistical learning and data mining. To achieve a better performance from our last version, we reconstruct most of our system architecture and add many modern algorithms to make better and better agent strategies. Most of our codes were refactored to reduce coupling between different Jobs and Tools. A roadmap-based dynamic partition algorithm was used to let our agents have much more powerful cooperation ability. Communication model is also updated, as we add an error-correction algorithm to increase the reliability.

ZJUBase has been devoted to Agent Competition since 2002. This year, a total of 5 students prepared for this competition. We divide our work into 5 parts: Police force agent strategy, Ambulance Team agent strategy, Fire Brigade strategy, communication and partition.

2. Software Architecture and Tools

Our architecture gains more flexibility, extensibility, testability and reliability, and reduce the coupling between components. This Software architecture, showed in Fig.1, consists of following main components:

- **JobBasedAgent**: Agent class derived from StandardAgent, including all necessary information used by our own strategy. Our main implementation is called “Job based”, which means we divided all agent action into several different jobs, an agent will choose one job to execute in each time step.
- **ScaleJob**: We use job to deal with different agent situation. Some jobs are common but others are designed for specific agent type. Fire brigades will use GetWaterJob, while police forces may use SaveOthersJob.
- **ToolKit**: Some jobs may use complex algorithms or information managers to assist their command decision, these are called tools. Some important tools are: CreateJigsawTool, CreatePartitionTool, HistoryTool, etc.

- **LiaisonManager**: We have designed a very tight communication model in case the channel situation is bad. This "Liaison" system encode every message into bit array and regroup them to bytes.

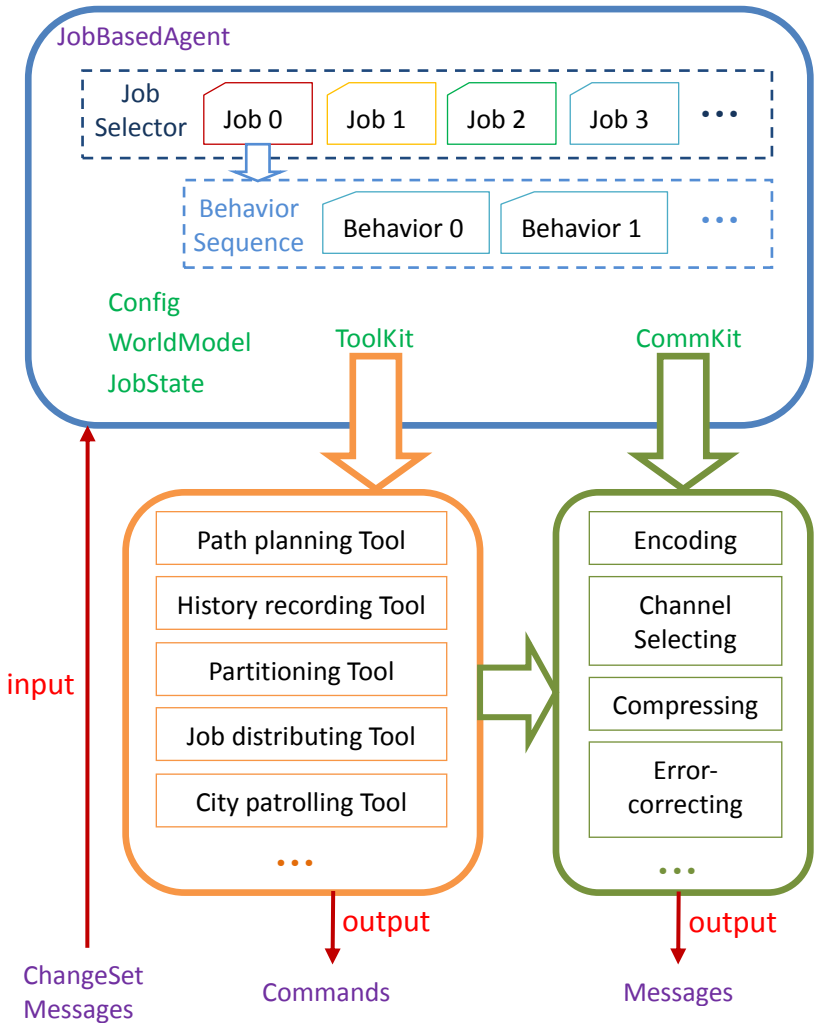

Figure 1 ZJUBase Software Architecture

3. Job selector

All of the agents are job based, which means we firstly search for all job spots that agents can do, then distribute them to each agent. And each agent has a list of jobs to do at a time. We divided the jobs into several types, **StuckJob**, **SaveSelfJob**, **SearchPartitionJob** and so on.

StuckInRoad/StuckInBuildingJob: These jobs have the highest priority, as the agent may be stuck in burning building or blockade. Generally, Agents will move smoothly, it will not choose path will too much blockade. But sometimes it may be trapped by the blockade on Road. To avoid this situation, we design **stuckInBuildingJob** and **stuckInRoadJob**. If agent thinks I am trapped, it will choose a point doesn't cover blockade, and try to get out of the trap. If failed for many times, agent will call PF for help.

SaveSelfJob: When agents get hurt, they will go to refuges if their damage is severe enough.

SearchPartitionJob: Agents will be assigned into different partitions so that we can sense the most changes of the whole world. When there is no specific jobs need to be done by an agent, it will start patrolling in its own partition. We also design a set of

algorithms to choose the best patrolling path, considering the importance of different buildings, partition condition and other information. For example, PF will check if there are roads I have never been to, neither have other PFs, if there is, PF will go to the farthest place.

Finally, the job selector will consider all the possible jobs and choose the most important one, and then the agent will execute several behaviors of that job. It may take several time steps to finish, and the current job may also be interrupted by other much more important job.

4. Message encode and error-correction

Communication condition varies between different maps, and it is vital for our agents to build a self-adaptive communication model which consists of every channel's real receiving rate detection and dynamic channel subscription and optimized message sending policy. Implementation of above features is detailed in the following.

Channel Assign. In the first cycle when agent commands are no longer ignored, agents begin to subscribe different channels and send detecting messages. In the second cycle, agents receive messages from the subscribed channels and analyze each subscribed channel's receiving rate. And to make this information fully shared, every agent will send the condition messages to every channel in the same way that detecting messages are sent. In the third cycle, agents collect condition messages and the total receiving rate can be calculated from these messages. Thus agents will have a real bandwidth for every channel which will be used in the channel assign process, and can use this information to assign different channel to different type of agents.

Encode & Decode. To shorten message length, messages are encoded and decoded on bit level rather than byte level. To further shorten the length of message, different properties are handled in different ways. For example, codec on damage and hp will take the perception precision into consideration; coding on agent id is the index of an agent's id order rather than the real EntityID. Every message is encoded with a head field followed by several ordered message property fields. To simplify the coding of messages, Java annotation is used to mark the order of message properties.

Hamming Code. To make this model more reliable, Hamming code [2] was also used. Its hamming distance is 3 so that it can detect and correct any one-bit error. It needs m extra parity bits to encode 2^m-m-1 data bits, which is the more efficient if our data bits number is larger. Below is a sample graph showing how hamming code works.

Bit position		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	…
Encoded data		P1	P2	D1	P4	D2	D3	D4	P8	D5	D6	D7	D8	D9	D10	D11	…
Parity Bit Coverage	P1	X		X		X		X		X		X		X		X	
	P2		X	X			X	X			X	X			X	X	
	P4				X	X	X	X					X	X	X	X	
	P8								X	X	X	X	X	X	X	X	

To check for errors, check all of the parity bits. If all parity bits are correct, there is no error Otherwise, the sum of the positions of the erroneous parity bits identifies the erroneous bit. For example, if the parity bits in positions 1, 2 and 8 indicate an error, then bit 1+2+8=11 is in error. If only one parity bit indicates an error, the parity bit itself is in error.

Actually, to gain a better performance, we use the extended hamming code system, which add another overall parity bit. The code can detect any two-bit error while correcting any one-bit error, or detect up to 3 errors if the decoder does not attempt to correct errors.

There are three kinds of noises in RCR system now: dropout noise, failure noise and static noise. The hamming code implementation can only solve the last one. For

other two noises, we will send some high priority messages several times to guarantee that the communication is reliable.

5. Roads classification

As there are usually so many roads in one map, it's hard for us for find ways to get some places or to find where the target is. To solve this question, we reduce the number of "roads" by classify them according to its position. We divide them into three different kinds. One is called the entrance, for this kind of roads are connected to the buildings, so if you'd like to go into the buildings you must pass these roads, the second one is called avenue, for they can be combined into a long street, the rest is the cross, these roads are connected to more than three roads. After the classification, we make a connection of them to form Jigsaw Avenues, Jigsaw Crosses. For different targets (like the buildings we may want to observe or extinguish, the roads that need cleaning), we firstly find out where they are, and then, we add them to their neighbor Jigsaw Avenues or Jigsaw Crosses (If the spot is connected to two Jigsaws, it's Ok, because we may choose the one which has a smaller Jigsaw number), finally, here come the Objective Jigsaw Avenues or the Objective Jigsaw Crosses. As there may be many jobs added to one Objective Jigsaw Avenue or one Objective Jigsaw Cross, it will be easy for our agents to finish all the jobs along one avenue or cross. It can save more time as we just need to discuss who is more fit for one Objective Jigsaw Avenue or one Objective Jigsaw Cross once and for all, on the other hand, after we finish all the jobs along those roads, we may let the agents check if there are other emergencies, like the other roads along the avenue may be blocked or there are other buildings that catch fire and we didn't notice beforehand, so these agents will still keep observing the avenue or cross to make sure there aren't more emergencies! The whole structure is like the flowchart below:

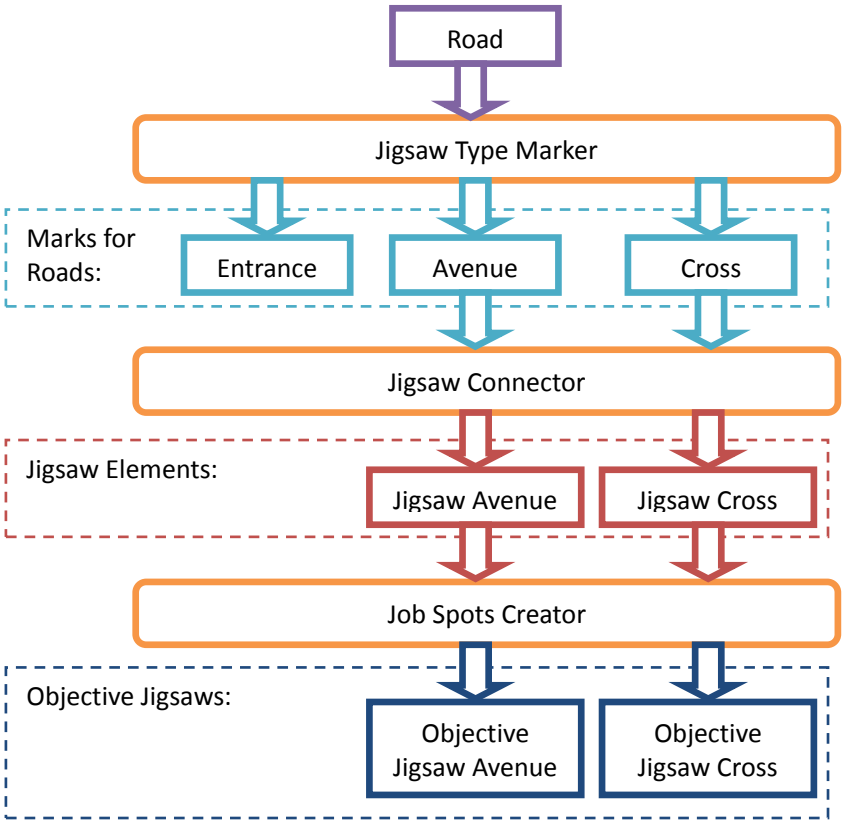

Figure 2 Jigsaw structure for road classification

These are viewers for our division strategy, crosses and avenues are correctly recognized, which can be used to speed up our path planning algorithm.

Figure 3 Road classification for Pairs (part) and VC maps

6. Partition Algorithm

As agents are distributed in the world and events around agents will be shared via communication, it is vital for agents to disperse so that more information can be gathered and agents can make better decisions.

Partition algorithm is mainly used to serve the task assignment, exploration and agent collaboration. As the performance of partition algorithm in our previous version is not very satisfactory, we implement another roadmap-based partition algorithm. It will satisfy following conditions:

- Every partition is a convex polygon, and the sharp is close to circle (not long and narrow).
- The boundary of each partition must be main roads.
- The variance between different partition's areas should be small.
- Partitions should not be too small. It must contain at least one circuit inside the partition, so that an agent can patrol along this path.

We create partitions on our Jigsaw layer, which is much higher than Area layer and it can also be ensured that connected buildings will be assigned to the same partition. As for partition number, now we use the square of the number of agents. This may be changed in the future. What's more, the number of agents may be changed when an agent died, so our partition implementation should also be changed smoothly, which means all agents can move little to switch to its new distributed partition. An example is shown in Fig.4.

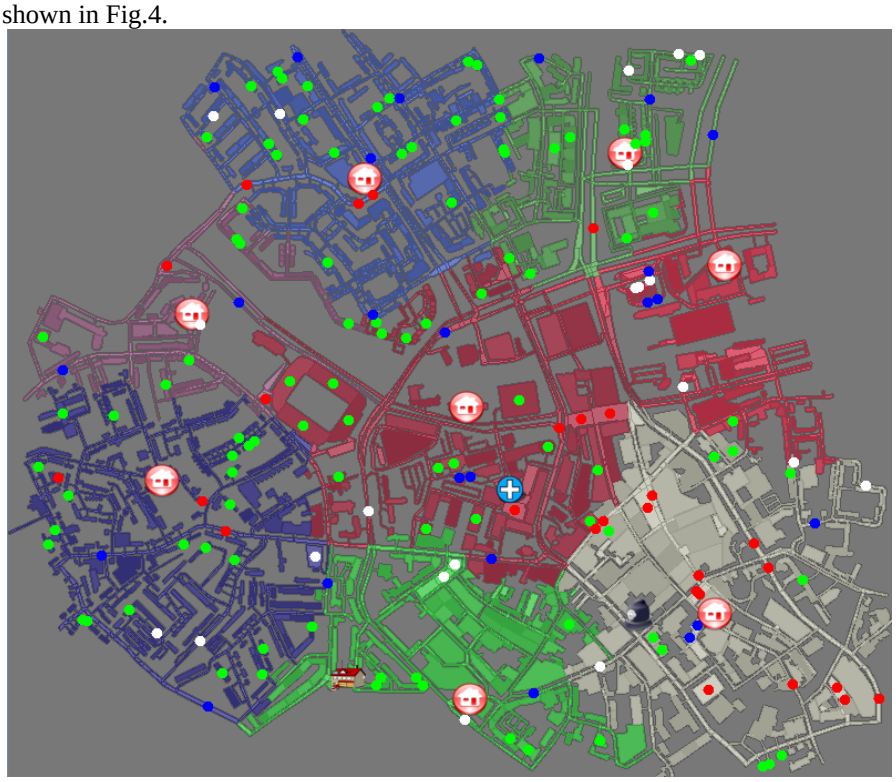

Figure 4 Map partition for Eindhoven

7. Agents

7.1 Ambulance team

We add a new system called 'rest in peace'. This system runs when ambulance team start rescuing agent. It is designed to judge the agent under rescuing can be rescued successfully or die during the rescuing. If ambulance team thought it has little possibility to rescue successfully, ambulance team will give up. This system skips those people who can't be rescued to save time for more valuable people. Because of what AT see (HP damage) are not accurate, but agent under rescuing himself can know

the exact value. So when AT is rescuing other agent, we make agent himself to tell the AT how long he will live to judge he is worth rescuing or not.

We also made a new cooperating strategy for ambulance teams. For each map, we calculate a number to limit the cooperation scale. Ambulance team will choose its target in a smarter way, so there won't be too many ambulance teams rescuing one target, ambulance teams can rescue as many people as possible.

7.2 Fire brigade

In our previous version we have already use many algorithms here, such as fire predictor and convex hull, now a pre-extinguishing strategy is also added.

Based on our powerful fire simulator component, we enhanced the function of estimating water quantity. Now our fire brigades are able to extinguish with minimized water required. Yet they are smarter on choosing target based on different situations. At the beginning of the whole procedure, we acquaint agents with the basic information and one set of most useful parameters, based on which they can estimate importance of each fired building and choose the best strategy. Besides, some tools are further optimized. In our previous version of convex hull, we decided the distance towards convex hull using center of hull, in this version we approach it using the rim of hull instead. And in fire partition, we now add consideration of key building (buildings located at the rim of block in connect with another block). This improvement helps to provide a more steady performance to fire brigades.

7.3 Police force

We optimize the priority of tasks for police forces and enhance the cooperation between ambulance team and police force. We also use new designed communication protocols to make them cooperate in a more efficient way.

We divide PFs' jobs into three levels of priorities, which are High, Mid and Low. In the jobs of high priority, there are saving ourselves (means going to the refuge to reduce damage), making sure the refuges can be access (means there aren't any blockades in front of the refuge, so that agents can get in there), also, there is one emergency we need to deal with, these are helping other agents when they are stuck by the blockades. Mid and Low level groups contain jobs like different kinds of exploration (for fire, for blocked roads and for civilians). In this version, we make the exploration saving more time, for we don't make the agents enter the room as what we did in our last version, instead, we just make sure, they can see what's in the building. In that way, we may save a few time steps so that PF agents can do more jobs.

About the new model of Police force clearing, we developed a new model for police force to check which point to clean. First of all, we divide the road into 3 parts: right part, middle part and left part. Then we will see if the blockade covers all the three parts, if so, then we can decide that the road isn't passable.

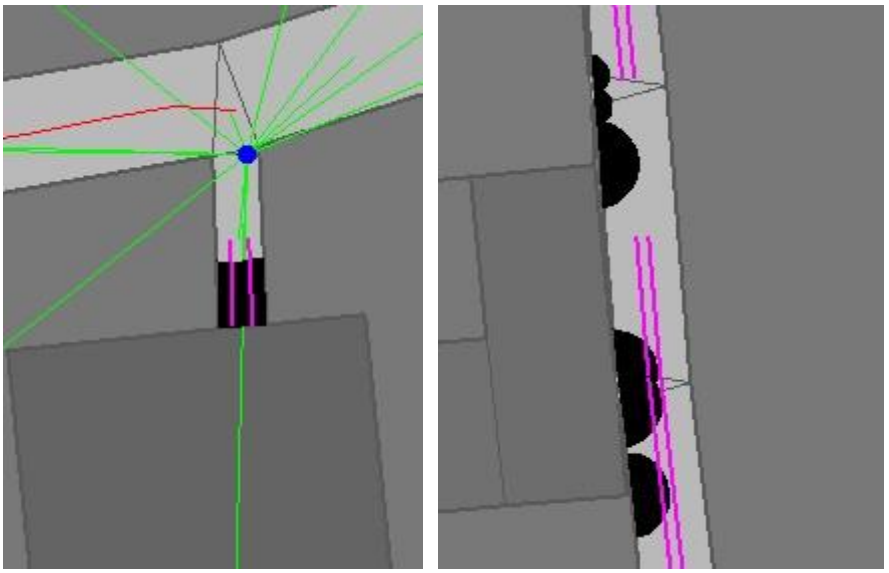

Figure 5 decide whether the road is passable
(left: impassable right: passable)

8. Acknowledgements

At last, we would like to thank all of the server developers' team for maintaining the rescue simulation server and for their technical support; thank RoboAKUT and SEU-Redsun and other leading teams for their source codes which inspire us a lot. Thank our seniors Yang Genmao, Feng Huan and Jin Yifan for their selfless help in many aspects.

9. References

[1] ZJUBase TDP 2012
[2] Hamming code – Wikipedia http://en.wikipedia.org/wiki/Hamming_code
[3] Visibility graph – de Berg, Mark, van Kreveld, Marc, Overmars, Mark, Schwarzkopf, Otfried, Computation- al Geometry, Springer-Verlag, 2000
[4] A* search – Hart, P. E., Nilsson, N. J., Raphael, B. "Correction to 'A Formal Basis for the Heuristic Determination of Minimum Cost Paths'". SIGART Newsletter, 1972, 37: 28–29
[5] Stefan-Boltzmann law – David Halliday, etc. Physics Wiley, 2001
[6] Jarvis march for convex hull – Cormen, Thomas H., Leiserson, Charles E., Rivest, Ronald L., Stein, Clifford. Introduc-tion to Algorithms. MIT Press and McGraw-Hill, 2001
[7] K-Means Clustering Algorithm – Kanungo, T., Mount, D. M., Netanyahu, N. S., Piatko, C. D., Silverman, R., Wu, A. Y. . IEEE Trans. Pattern Analysis and Machine Intelligence, 2002, 24: 881–892