RoboCup 2017
Rescue Simulation League Team Description
RoboAKUT (Turkey)

H. Levent Akın, Okan Aşık

Boğaziçi University, Turkey
[akin@boun.edu.tr, okan.asik@boun.edu.tr]

Abstract

RoboAKUT is a multi-agent rescue simulation team that competes in RoboCup Rescue Agent Simulation League(RSL). It has been competing in RoboCup competitions since 2002 and won the *First Place* in the agent competition in RoboCup 2010 and in RoboCup IranOpen 2017. RoboAKUT code base is rewritten to use Agent Development Framework(ADF) ADF requires the development of target selectors and action modules which calculates the RSL simulator actions for the selected targets. Also, these modules uses basic algorithms such as path planning, clustering, and exploration. We develop reward-based task allocation mechanism where we calculate utility for targets. The utility values are calculated as the expected total reward using value iteration algorithm. RoboAKUT team got the first title in RoboCup IranOpen 2017.

1 Team Members

- H. Levent Akın (Advisor)
- Okan Aşık (Developer)

2 The Agent Development Framework

Starting from RoboCup 2017, all the competing teams are required to use Agent Development Framework (ADF). The new RSL teams generally struggle to develop a fully working agent framework because of the complexity of the RSL simulator. The ADF aims to provide a simple interface for the agent development by encapsulating the details of the RSL simulator. The behavior of an agent is determined by its *Tactic*. The target selection and the calculation of an action for the selected target are coordinated by *Tactic* module. The module also manages messages that inform all modules about the new state of the world that is perceived by the agent. The competition prohibits to change the *Tactic* so that all teams compete with the same *Sample Tactics* provided by ADF. There is a *Tactic* module for each agent type as follows:

- **Mobile Agents:** There are three types of mobile agents in RSL; *Ambulance Team*, *Fire Brigade*, and *Police Force*. Every agent carries out a specific task. The RSL simulator runs a single time step, sends perception messages for agents

and waits for an action from agents. The *Tactic* updates world view of all the modules with the new perception, calculates an action, and finally sends the action message to the simulator. All mobile agents use a *target detector* and a *search* module. The *target detector* determines a target according to agent type and if the target detector cannot detect a target, the *search* module selects a target for the purpose of exploration. Once a target is selected either by *target detector* or *search* module, the *action* modules calculate an RSL action according to the selected target. However, *move action* module is common for all mobile agents because when the agent chooses an exploration target, its action is calculated by *move action*.

  - **AmbulanceTeamTactic** uses *human detector* and *search* modules for the human (either a civilian or a mobile agent) target selection. The *transport action* and *move action* calculate a simulator action for the selected target. The *transport action* action chooses either to transport, rescue, or move to the target. When the *Ambulance Center* assigns a target for the agent *ambulance command executor* modules are used.
  - **FireBrigadeTactic** uses *building detector* and *search* modules for the building target selection. The *firefighting action* and *move action* calculate a simulator action for the selected target. For example, if the building to extinguish is not close enough to extinguish, the agent chooses to move towards the building. When the *Fire Station* assigns a target for the agent *fire command executor* modules are used.
  - **PoliceForceTactic** uses *road detector* and *search* modules for the road target selection. The *clear action* and *move action* calculate a simulator action for the selected target. For example, if there are blockades on the path to the selected road target, they are also cleared by the agent. When the *Police Station* assigns a target for the agent *police command executor* modules are used.

- **Center Agents:** There are three center agents in RSL; *Ambulance Center*, *Fire Station*, and *Police Office*. The center agents act as a communication center when scenario permits a radio communication. ADF provides a simple communication framework on RSL. The teams can use this communication framework to develop their strategies to allocate tasks to the agents. However, center agents do not know the tasks since they cannot perceive the world as mobile agents do. Therefore, they create a task list by using task messages sent by the mobile agents. This task list aggregation mechanism is already provided by ADF.

  - **AmbulanceCenterTactic** uses *human target allocator* and *human command picker* modules to assign human targets to the agents. The *human target allocator* assigns the targets to the closest mobile agents and the *human command picker* creates a command message for the agent.

- **FireStationTactic** uses *fire target allocator* and *fire command picker* modules to assign buildings that are on fire to the agents. The *fire target allocator* assigns all targets to the closest mobile agent and the *fire command picker* creates a command message.
- **PoliceOfficeTactic** uses *road target allocator* and *road command picker* modules to assign blocked roads to the closest mobile agents. The *road target allocator* assigns all blocked road to the closest mobile agent and the *road command picker* creates a command message.

3 RoboAKUT Strategies

RoboAKUT extends *target detectors*, *actions*, *path planning*, *clustering* and *search* modules of ADF.

3.1 Target Detectors

The main goal of the target detectors is the task allocation. The module chooses a task from available tasks based on the limited world view of the agent. For example, a *Fire Brigade* agent needs to choose a building to extinguish from the list of buildings that are on fire. Every agent type has its own target detectors. *Police Force* agent uses *road detector*, *Fire Brigade* agent uses *building detector*, and *Ambulance Team* uses *human detector*.

3.1.1 Road Detector

The road detector chooses a *clearing* target with a decision tree that is tuned by experiments. The decision tree implements a priority based targets. At every decision level, the agent checks the condition for the behavior of that level. For example, if the agent perceives a human that is stuck in a blockade, clearing this blockade has the highest priority. The behaviors of the *road detector* is as follows starting from the highest priority:

1. **Clearing for a Human:** The agent looks for humans (civilians or mobile agents) in its world model. For every human, the agent checks whether the human is stuck and alive. A human can stuck in two ways; it is either just inside a blockade (a polygon inside a road polygon), or a blockade is just in front of the building where the human stays in. The agent chooses the closest of the stuck humans if there are more than one stuck human.

2. **Clearing the Cluster:** If there is no agent to rescue in the world model of the agent, the agent will clear the roads of its own cluster. Every agent has an assigned cluster. The clustering algorithm assigns a cluster to every agent based on the distance of the agent to the center of the clusters. The algorithm also ensures that every cluster constructs a connected graph. The agent keeps track of

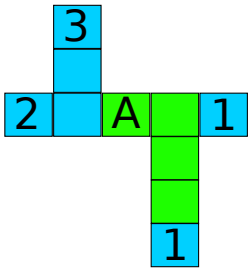

Figure 1: An example illustration of cluster clearing behavior. The blue and green squares represents the roads in the cluster. The visited ones colored as green. *A* represents the agent position and values at the leaf nodes show the utility values calculated as the sum of unvisited roads on the path.

the visited roads. The ultimate aim of this behavior is to maximize the number of cleared roads. Therefore, we formalize the problem as a task allocation where the tasks are the *leaf* nodes of the cluster graph. We define the *leaf* as the node having only one edge. The utility of a *leaf* node is calculated as the number of the *unvisited* roads that agent needs to take to reach the *leaf* node. The path to a *leaf* node is calculated using a shortest path algorithm. The behavior chooses a *leaf* node having the highest utility value (see Figure 1)

3.1.2 Building Detector

The building detector algorithm chooses building targets in two ways: *reactive planning* and *general allocation*. In *reactive planning*, the module gets all the building in extinguish range that are on fire and chooses randomly one of them. Firstly, in our experiments, we see that the reactive behaviors are quite effective despite their simplicity. Secondly, since we do not coordinate the firefighting agents, we reduce the resource inefficiency due to error in allocations. Random selection ensures the equal distribution of agents for a given set of targets. The algorithm is concurrently running for every agent and agents may not know the current position of other agents. If an agent does not get any message or does not perceive other agent, it uses the last position of other agents. If the agent does not have information about the other agents after the start of the simulation, it uses the initial position of the agent. Although this looks like an ineffective method, since agents generally moves around their starting position, it becomes a good approximation for the position of the agent.

If there is no building in extinguish range, the module uses *general allocation* and assigns buildings on fire to all the agents based on the distance between the building and the agent. The algorithm iterates over the buildings on fire and gets two closest agents to the building at every iteration. If this is the agent that is running the algorithm, it is assigned to the current building and iteration ends. If the agent is not one of two agents, these agents are removed from agent list, and the algorithm continues with the next building.

3.1.3 Human Detector

The human needs to be rescued when it is buried or wounded. If a human is buried, the agent needs to rescue the agent from the debris. If a human is wounded, the agent needs to transport the human to the refugee. The human detector module assigns humans that needs to be rescued or transported to the agents. This module makes a central assignment similar to the building detector based on the distance of the agents to the humans which needs to be rescued.

3.2 Actions

The *action* modules calculate a simulator action for mobile agents according to the selected target. There are four *action* modules; *move action* is used by all mobile agents, *clear action* is used by *Police Force*, *transport action* is used by *Ambulance Team*, and *extinguish action* is used by *Fire Brigade* agents.

3.2.1 Move Action

The module calculates a shortest path between the position of the agent and the given destination. The path is the list of areas (roads and buildings) on the map. The simulator accepts the path as a movement command.

3.2.2 Clear Action

The module gets a target position to clear. However, that requires the clearing of the blockades that are on the path to the target. Therefore, as the first step of the algorithm, the shortest path between the agent and the target position is calculated. The simulator clear action is a vector starting from the position of the agent. The module creates clear vector towards the next position on the shortest path. If this clear vector intersects with a blockade, the module needs to clear this blockade. If the clear vector does not intersects, a new clear vector created from the end point of the previous one. This processes continues until the clear vector reaches the end of the path. If the clear vector intersects with a blockade, the module checks whether the agent can clear the intersected blockade. If the agent is able to clear the blockade, a clear simulator action created and sent by the module. If the agent is not able to clear the blockade, a simulator move action is created and sent by the module.

If the target selector module chooses a position to rescue a human from blockade, the clear action needs to calculate a clear vector towards the position of the human. By design, the target selector can choose only the areas to clear, therefore the clear action needs to calculate the position of the human from selected position.

3.2.3 Transport Action

The transport action calculates a simulator action for the selected human. If the target human and the agent is not on the same position (road or building), the module calculates

a shortest path to move to the position of the target. Once the agent and the target is on the same position, the module creates and sends a rescue action if the human is buried and the module creates and sends a load action if the human is not buried and is wounded. After the human is loaded to the agent, it is transported to the shortest refugee and unloaded there.

3.2.4 Extinguish Action

The extinguish action calculates a simulator action for the selected building that is on fire. If the agent needs to refill its tank, the module calculates a shortest path to the closest refugee. If the agent has water, it creates and sends an extinguish simulator action for the target building. If the building is not on the extinguish range, the agent creates and sends a move simulator action by calculating the shortest path to the target building.

3.3 Path Planning

The path planning algorithm calculates the shortest path between two positions on the map. The map of RSL simulator consists of *roads* and *buildings*. They construct a connected graph where the edge between vertices are defined as the neighbors. The distance between neighbor *roads* and *buildings* are calculated as the distance between the centers of these areas. We use Dijasktra's shortest path algorithm [**?**] to calculate the shortest path between two areas. Dijasktra's algorithm starts from the start vertex and builds a path using the shortest path from possible neighbors. The complexity of the algorithm is $O(|E| + |V|^2)$, where E stands for the number of edges (neighbors) and V stands for the number of vertices (areas). At every iteration the algorithm chooses a vertex having the shortest path starting from the start vertex. By implementing the list which keeps the paths sorted according to their distance as the priority heap, we can reduce the complexity to $O(|E| + |V|log(|V|))$.

We also keep track of blocked roads and temporarily do not calculate them as a neighbor for T time steps only for *Ambulance Team* and *Fire Brigade* agents. As the agent moves on the map, the perceived roads are updated. This ensures the calculation of a path that are clear of blockades. After T time steps, the blocked roads are reset and assumed to be clear of blockades until the agent discovers that the road is still blocked.

3.4 Clustering

The clustering algorithm is used to create a hierarchical task allocation for the agents. By assigning every agent to a particular area (roads and buildings), we aim to achieve a balanced distribution of the agents on the map. We have building clustering algorithm for *Fire Brigade* and *Ambulance Team* agents and road clustering algorithm for *Police Force* agents.

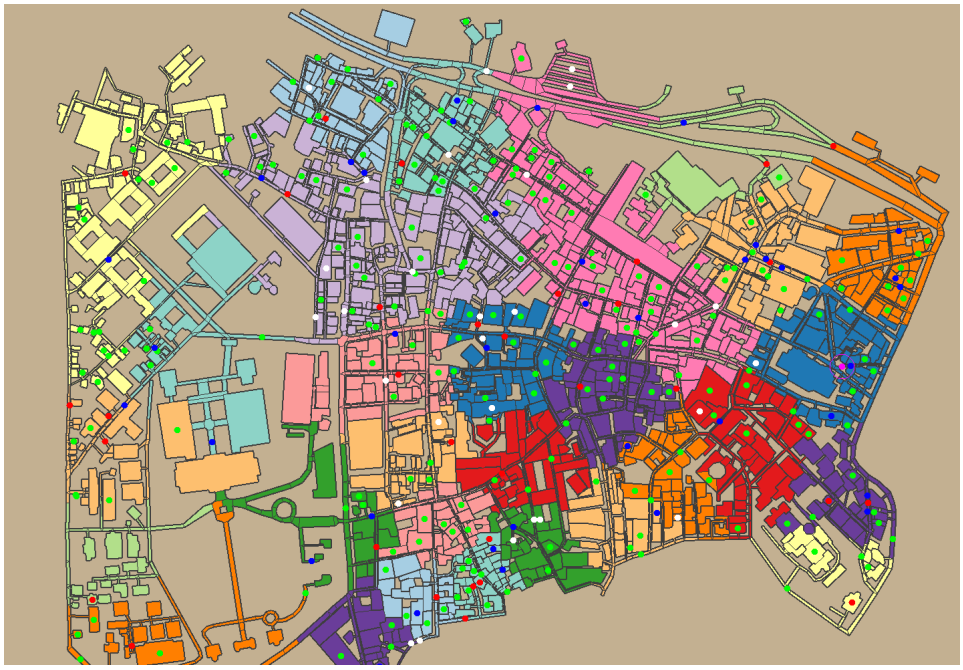

Figure 2: An example visualization of building clustering. The different colors represent different clusters, but if two clusters with the same cluster becomes side to side they seem like a single cluster (such as pink colored cluster).

3.4.1 Building Clustering

The buildings are clustered by k-means clustering algorithm [1]. We set the number of clusters same as the number of agents that are same type, such as *Fire Brigade* agents. Firstly, the algorithm sets random cluster centers. Then, it iterates over the buildings and adds them to the closest cluster based on the air distance to the cluster centers. Based on the member of the cluster a new cluster center is calculated. Finally, this process continues for a given number of iterations. An example visualization of building clustering can be seen in Figure 2.

3.4.2 Road Clustering

We would like to cluster the roads as a connected graph. Since the *Police Force* agents needs to clear roads, they need to move on the connected roads. Therefore, if the road cluster is not a connected graph, the agent needs to move on the other agents clusters. Also, we use task allocation scheme using leaf of the graph as explained in Section 3.5.

We could use k-means algorithm, but this would result in disconnected graph since the distances are measured as air distance. Also, some member of the clusters could be close by air distance by might be very far away by movement distance (distance that is needed to be covered by the agent). Therefore, we need to measure the distance between the cluster center and the road by movement distance. However, this requires

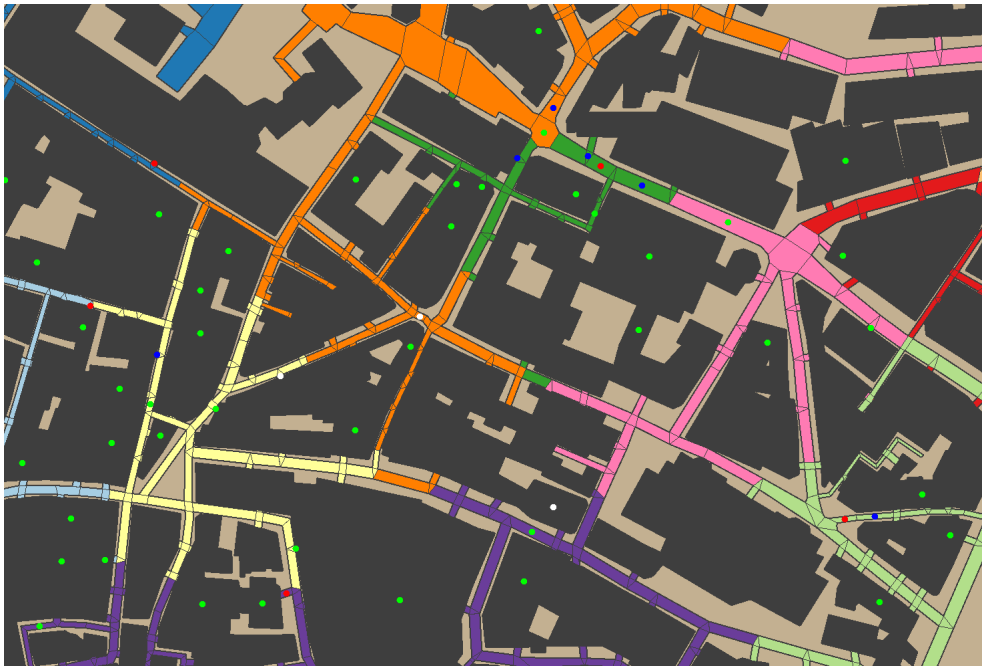

Figure 3: An example visualization of road clustering. The different colors represent different clusters. These clusters create a connected graph so that some roads may be member of more than one clusters.

the shortest path calculation at the assignment of every road. This algorithm performs very slow compared to air distance calculation. Therefore, we propose to use air distance, but after k-means clustering, we use an algorithm to connect the disconnected graphs of the cluster by the shortest path. Finally, the roads on the connecting shortest paths are also added to the clusters. An example visualization of road clustering can be seen in Figure 3.

3.5 Search

The search module achieves the exploration for *Fire Brigade* and *Ambulance Team* agents since the road detector module already implements an exploration algorithm. (see Section) The module keeps a list of buildings to search, and removes the perceived buildings from the list. There are two lists; all buildings list and the list of buildings only from the cluster of the agent. The algorithm chooses the closest building from one of the list with equal probability. Since the agents choose targets based on the distance, this ensures that agents check the buildings for a victim or a fire one by one. This achieves the full map exploration because we want to extinguish a fire as quick as possible. If there is a fire just started at some corner of the map, all the *Fire Brigade* agents needs to act on it. Otherwise, this fire may spread all over the map and becomes impossible to extinguish. The lists are also reset after T time steps because the building that is not fire may catch a fire in the future.

Table 1: The RoboCup IranOpen 2017 competition scores. S stands for the simulator score and P stands for relative scoring that is defined in competition rulebook. R shows the rank of the teams according to total points.

Preliminary Round															
R	Teams	Kobe1		Paris1		Istanbul1		Eindhoven1		VC1		Kobe2		Total	
		S	P	S	P	S	P	S	P	S	P	S	P	S	P
1	MRL	84.48	18.0	157.79	12.0	125.65	24.0	105.75	24.0	84.5	24.0	60.71	15.0	618.88	117.0
2	**RoboAKUT**	87.52	24.0	164.26	14.0	116.69	15.0	84.88	11.0	72.31	19.0	99.81	24.0	625.47	107.0
3	Allameh Tab.	84.52	21.0	152.15	9.0	112.8	12.0	100.25	20.0	79.93	22.0	41.03	10.0	570.68	94.0
4	ApolloRescue	52.76	3.0	206.36	24.0	121.69	20.0	99.38	19.0	43.34	6.0	70.58	17.0	594.11	89.0
5	A.T.F	84.48	19.0	147.75	6.0	108.42	7.0	96.23	18.0	70.14	18.0	37.95	8.0	544.97	76.0
6	IR-Force	84.52	20.0	144.13	5.0	111.7	11.0	80.09	9.0	49.25	9.0	41.02	9.0	510.7	63.0
7	AAI	86.54	23.0	154.7	11.0	106.58	5.0	86.49	12.0	54.79	11.0	0.0	1.0	489.11	63.0
8	S.O.S	84.47	17.0	144.02	4.0	111.26	10.0	84.7	10.0	43.64	7.0	52.25	13.0	520.33	61.0
9	Aura	86.51	22.0	148.34	8.0	107.51	6.0	72.23	4.0	56.46	12.0	37.52	7.0	508.56	59.0
10	LarvicSaurus	28.23	1.0	154.17	10.0	110.56	9.0	90.97	15.0	48.66	8.0	60.03	14.0	492.62	57.0
11	SEU-UniRobot	23.65	1.0	148.29	7.0	110.29	8.0	60.89	1.0	27.45	1.0	29.09	6.0	399.66	24.0
12	Roshd	17.99	1.0	136.87	3.0	108.42	7.0	58.7	1.0	27.06	1.0	17.41	5.0	366.44	18.0
Semi Final															
R	Teams	Istanbul2		Paris2		Mexico1		Berlin1		VC2		Kobe3		Total	
1	ApolloRescue	150.98	6.0	126.44	7.0	203.45	11.0	102.34	16.0	50.86	14.0	91.33	16.0	725.4	70.0
2	Allameh Tab.	159.93	8.0	117.18	3.0	211.48	13.0	98.16	13.0	45.29	10.0	81.39	8.0	713.43	55.0
3	**RoboAKUT**	188.25	16.0	123.7	6.0	164.94	4.0	72.34	1.0	52.29	16.0	84.99	11.0	686.51	54.0
4	A.T.F	187.25	15.0	141.88	14.0	130.95	1.0	82.47	4.0	51.09	15.0	77.11	5.0	670.75	54.0
5	MRL	169.37	10.0	121.06	5.0	171.85	6.0	91.87	9.0	41.31	7.0	81.32	7.0	676.77	44.0
6	Aura	168.48	9.0	146.11	16.0	136.4	1.0	79.84	3.0	36.22	3.0	82.05	9.0	649.09	41.0
7	S.O.S	112.75	1.0	128.49	8.0	210.47	12.0	93.21	10.0	35.92	2.0	74.97	4.0	655.81	37.0
8	IR-Force	128.53	1.0	111.29	1.0	224.55	16.0	86.0	6.0	24.95	1.0	67.68	1.0	643.0	26.0
Final															
R	Teams	Joao1		Eindhoven2		Mexico3		NY1		Berlin2		Istanbul3		Total	
1	**RoboAKUT**	96.73	1.0	129.54	8.0	86.57	8.0	69.96	8.0	179.93	7.0	100.32	4.0	663.05	36.0
2	A.T.F	132.37	8.0	121.51	1.0	77.9	6.0	41.27	2.0	174.19	6.0	74.23	1.0	621.47	24.0
3	ApolloRescue	124.96	6.0	126.5	5.0	50.32	1.0	41.24	1.0	180.2	8.0	78.38	1.0	601.61	22.0
4	Allameh Tab.	92.13	1.0	122.3	1.0	56.09	1.0	43.99	3.0	127.32	1.0	128.1	8.0	569.94	15.0

4 RoboCup IranOpen 2017 Results

The performance of RoboAKUT in RoboCup IranOpen 2017 competition can be seen in Table 1. RoboAKUT completed the preliminary round as the second, semi final as the third, and win the competition by completing the final round as the first team. The relative scoring punishes team if they are good at certain scenarios and worst in other ones. The best team of the scenario gets the highest score and the worst team gets 1. Therefore, relative scoring enforces robust performances. RoboAKUT becomes the worst team in final scenarios only once, becomes the best team three times.

References

[1] Ethem Alpaydin. *Introduction to machine learning.* MIT press, 2014.