RoboCup Rescue Simulation Machine Learning Workshop

Arnoud Visser[1], Luis Gustavo Nardin[2], and Sebastian Castro[3]

[1] Universiteit van Amsterdam, Amsterdam, The Netherlands
[2] Brandenburg University of Technology, Cottbus, Germany
[3] MathWorks, Natick, Massachusetts

Abstract. The challenge of the Rescue Simulation League is to learn an optimal response for a team of robots that has to mitigate the effects of a natural disaster. To operate optimally, several problems have to be jointly solved, such as task allocation, path planning, and team formation. Solve these difficult problems can be quite overwhelming for newcomer teams. We create a workshop that demonstrates how these problems can be approached using standard artificial intelligence and machine learning algorithms available in MathWorks® Statistics and Machine Learning Toolbox™. We demonstrate how to use this toolbox (1) to analyze and model disaster scenario data for developing more elaborated rescue decision-making algorithms, and (2) to incorporate state-of-the-art machine learning algorithms into RoboCup Rescue competition code using the MATLAB® Engine API for Java.

1 Introduction

Urban Search and Rescue (USAR) scenarios offer a great potential to inspire and drive research in multi-agent and multi-robot systems. Since the circumstances during real USAR missions are extraordinarily challenging [1], benchmarks based on them, such as the RoboCup Rescue competitions, are ideal for assessing the capabilities of intelligent robots. The goal of the RoboCup Rescue competitions is to compare the performance of different algorithms for coordinating and controlling teams of either robots or agents performing disaster mitigation.

The rescue agent simulation competition aims to simulate large scale natural disasters, such as earthquakes, and explore new ways of autonomous coordination of heterogeneous rescue teams under adverse conditions. This competition was first demonstrated during RoboCup 2000 [2] and was later launched as an official competition in 2001. The teams participating in the agent simulation competition have their background mainly from artificial intelligence and robotics research.

The competition is based on a complex simulation platform representing a city after an earthquake (Figure 1). In the competition, fire brigade, police force and ambulance team agents extinguish fires, unblock roads, and rescue victims trapped inside collapsed buildings, respectively. The team scoring is based on the number of victims rescued and the number of remaining buildings with

Fig. 1. Impressions of the paths for the agents inside the central sector of Kobe city [3].

various levels of fire damage. The urban areas considered in this competition typically contain up to 5000 buildings and teams of fire brigades, police forces and ambulance teams may not consist of more than 50 agents. There may be up to 1000 civilians that need to be rescued.

The complexity of this rescue scenario imposes several challenges to the development of different aspects of multi-agent systems like task allocation with uncertainty, coalition formation, cooperation, distributed control, and communication. Artificial intelligence, and in particular machine learning methods are very well suited to cope with some of these challenges. For instance, fire brigades can optimize their decisions by estimating buildings' danger of fire ignition (discrete state – classification) and ambulance teams can optimize their rescue operations by predicting more accurately the chance of potential victims to survive (continues state – regression).

This workshop aims to instruct the participants:

1. how to use MATLAB® and add-on packages such as the Statistics and Machine Learning Toolbox™ to analyze and model disaster scenario data using both interactive design tools (GUIs) and by writing MATLAB® code. This analysis will provide support to the development of more data-driven and elaborate rescue decision-making algorithms (see Section 2).
2. how state-of-the-art machine learning algorithms can be directly incorporated into their competition software framework using the MATLAB® Engine API for Java (see Section 3).

2 Interactive approach

MATLAB®, Statistics and Machine Learning Toolbox™, and other add-ons can be used in interactive mode in different situations.

2.1 Unsupervised methods

These tools can be used to analyze and model disaster scenario data with unsupervised machine learning methods. Clustering algorithms are interesting for rescue simulation teams to partition the map into sectors and evenly distribute the search and rescue workload among agents [4, 5]. Available clustering algorithms in MATLAB®[4] include k-means [6], k-medoids [7], hierarchical clustering [8], Gaussian mixture models [9], and hidden Markov models [10]. Figure 2 shows the partitioning of the buildings in the Paris map using k-means clustering algorithm based on buildings location.

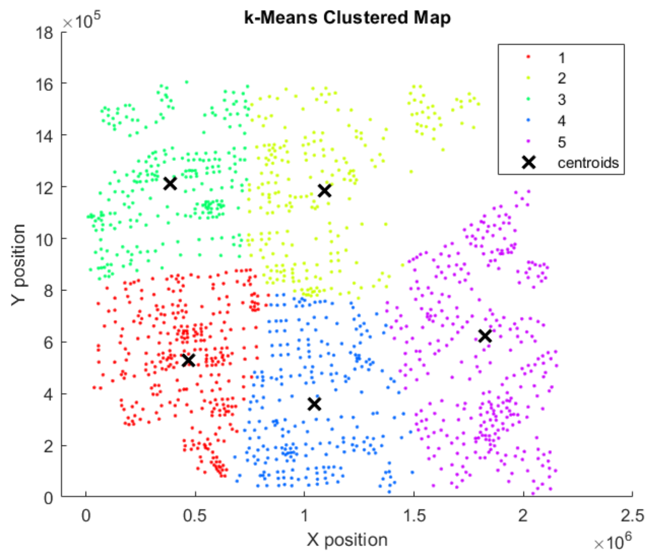

Fig. 2. k-means Clustering of building locations with MATLAB®.

2.2 Supervised methods

Analysis of the disaster scenario using supervised machine learning methods can be applied to learn associations between observable variables and hidden variables (part of a causal model of the world). Estimates of the value of variables of a world model could be done with discrete states (classification) or continuous states (regression). An example of a classification that would be of interest to optimize the decision of fire brigades is the classification of buildings based on the danger of fire ignition. A useful usage of regression for the ambulances would

[4] `https://www.mathworks.com/products/statistics/features.html`

be to predict the chance victims have to survive based on an estimate of the remaining health points (HP) at the end of the scenario simulation.

Statistics and Machine Learning Toolbox™ has several algorithms for classification and regression[5]. One can choose for classification between methods like boosted decision trees, naive Bayes classifiers, K-nearest neighbors or support vector machines. Figure 3 shows the K-nearest neighbors (KNN) classification used to predict if a civilian would be dead, in a critical state, injured or in a stable state at the end of the scenario simulation. This classification predicts correctly 78.9% of the civilians' state. However, notice that most of the wrong detections (i.e., sum of the numbers in the red cells) are above the diagonal green cells meaning that this trained classifier predicts a civilian in a less severe state than the civilian really will be. For instance, there are 9 cases in which the civilian will die and the classifier predicted it as injured.

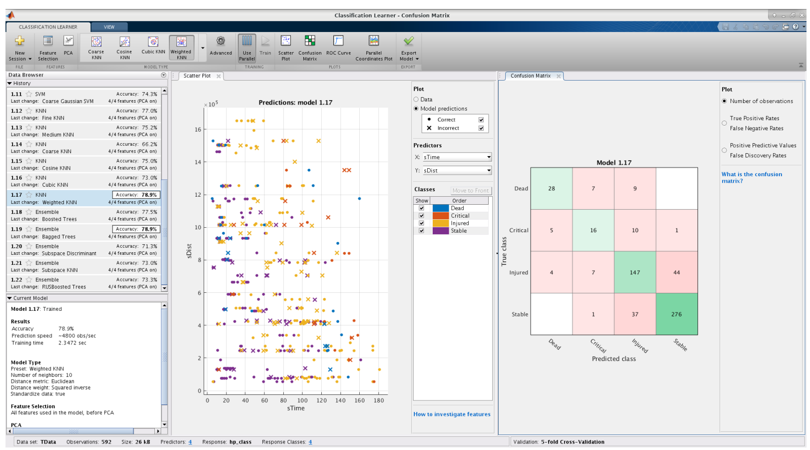

Fig. 3. Classification Learner MATLAB® app showing predictions of the injury class of the civilians at the end of the scenario using Weighted K-Nearest Neighbors.

In Statistics and Machine Learning Toolbox™, data can be preprocessed with dimensionality reduction methods like principal component analysis and singular value decomposition, followed by linear or non-linear regression methods. The results can be visualized with ensembles like random forests, boosted and bagged regression trees. To learn those ensembles several optimization algorithms like AdaBoost and TotalBoost are available. Figure 4 shows an example of an ensemble fit into a bagged tree model with the estimate of the remaining health points (HP) at the end of the simulation scenario with a root mean square error (RMSE) between the predicted and true amount of HP equals 1167.5 (and normalized RMSE equals 0.1228).

[5] `https://www.mathworks.com/discovery/supervised-learning.html`

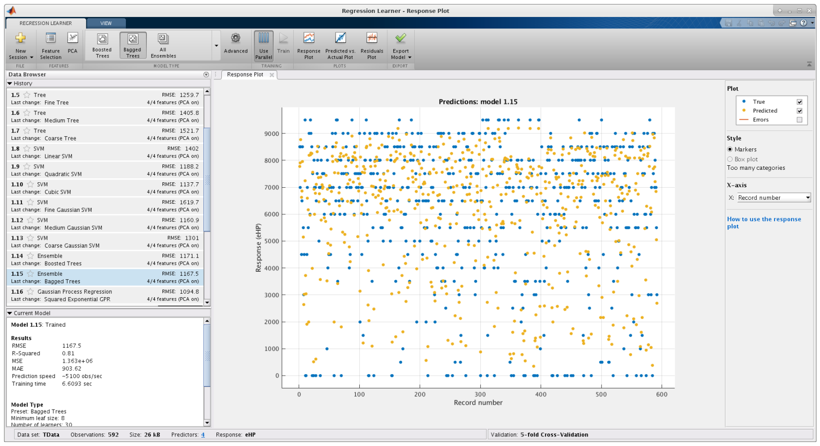

Fig. 4. Regression Learner MATLAB® app showing predictions of the chance to survive (remaining HP) of civilians buried in the devastated city.

2.3 Path planning

If an agent wants to move to a specific location to perform a task, a path plan to that location has to be defined. Two possible approaches to tackle this problem are (1) to use the path planning algorithms from the MATLAB® graph and network algorithms[6] or (2) to use the graph-routines from Peter Corke's Robotics Toolbox, which is part of his textbook [11].

First, however, all the roads of a city map needs to be converted to a graph in MATLAB® format. The nodes of the graph are identified by the roads ID and they also store the actual (x, y) location of the road to facilitate the visualization of the results (see Fig. 5). The Java code to generate such a MATLAB® graph can be called during the precompute phase of the competition, and its pseudo-code is:

```
For (Entity next: this.worldInfo.getEntities() ) {
  loc = this.worldInfo.getLocation(next.getID());
  matlab.eval("G=addnode(G,table(next.getID(),loc.first(),loc.second());");
}
For (Entity next: this.worldInfo.getEntities() )
  Collection areaNeighbours = next.getNeighbours();

  for(entityID neighbour : areaNeigbours) {
    matlab.eval("G=addedge(G,find(next.getID()),find(neighbour.getID());");
  }
}
matlab.eval("save('graph.mat',G);");
```

Once created, the graph in MATLAB® can be queried, for instance to get the shortest path between two nodes. This can be done by calling a MATLAB® script which contains the function `short_path = getPath(from,targets)`, which not only loads the graph `G`, calls the MATLAB® method `[TR,D]=shortestpathtree`

[6] `https://www.mathworks.com/help/matlab/graph-and-network-algorithms.html`

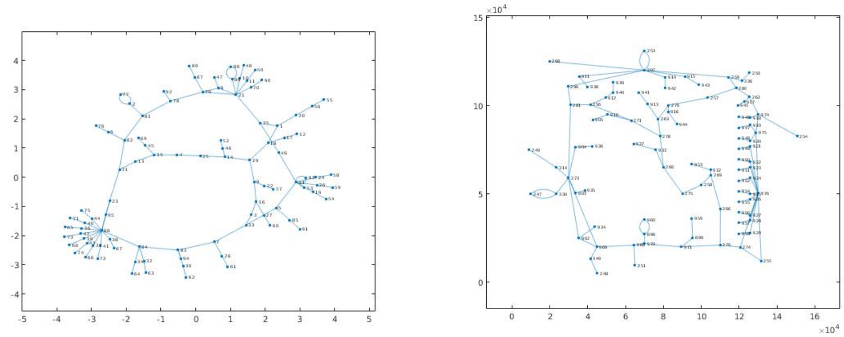

Fig. 5. The small Test map of the RoboCup Rescue agent competition, in MATLAB® as topological (left) and metrical graph (right).

and sorts the resulting paths `TR` based on the distance `D`. It is possible to specify in MATLAB® the algorithm to use (Breadth-first or Dijkstra). It is also possible to use A*, this algorithm is available in Peter Corke's robotics toolbox [11]. The MATLAB® code of this algorithm is open source and well documented. This makes it possible to modify the A* algorithm to Dijkstra's algorithm (by removing the heuristics) or breadth-first (by not sorting the frontier on distance so far). The only thing needed is a script to translate from MATLAB® native `graph`-format to Peter Corke's `Pgraph`-format. For smaller competition maps like Kobe this can be done in 13 seconds (measurement with a computer with a Core i7-8550U processor), for larger competition maps like Paris 22 seconds are needed for this conversion (see Fig. 6). This is fast enough for the precompute phase of the competition.

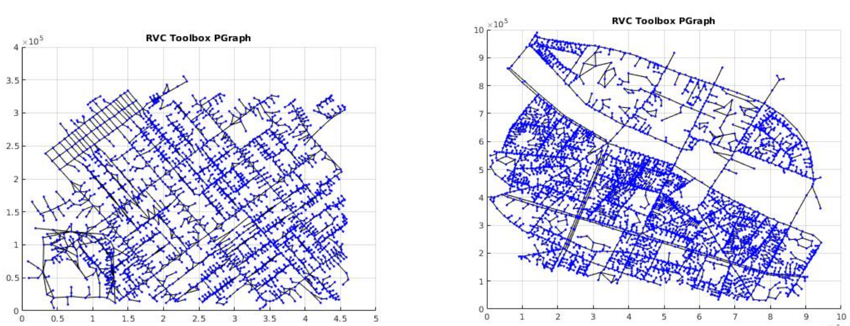

Fig. 6. The RoboCup Rescue agent competition maps of Kobe (left) and Paris (right) in Peter Corke's `Pgraph`-format.

An advantage of this approach compared to the path-planning methods typically applied by the RoboCup Rescue teams is that each agent can load this *a priori* map and modify the edges based on the blockades observed and/or communicated. This information can even be updated when police force agents clear part of the road.

2.4 ROS interface

When an agent has reached a building, it has to enter this building. This challenge is part of the Virtual Robot competition [12]. The MATLAB®, Robotics System Toolbox makes it possible to directly control robots and realistic simulation via the Robotics Operating System (ROS) interface, as demonstrated in the Future of RoboCup Rescue workshop [13] and the more recent RoboCup@Home Education workshop[7].

Inside the Virtual Robot competition one of the hard challenges is to detect buried victims from the camera images. In the workshop victim detection could be demonstrated with the MATLAB® deep learning capabilities, a combination of the Neural Network Toolbox, Parallel Computing Toolbox, GPU Coder, and Computer Vision System Toolbox. Note that MATLAB® can run models deployed to a GPU faster than TensorFlow or Caffe, which is highly desirable for robotic applications[8].

3 ADF integration

In addition to benefit of the MATLAB® models and algorithms using the interactive design tools, these models and algorithms can also be integrated with the Agent Development Framework (ADF) [14]. ADF is an agent architecture whose use is mandatory to all teams in the rescue agent simulation competition. This agent architecture is composed of several, highly specialized modules responsible for different data processing and decision-making tasks, such as clustering, path planning and task allocation. In the workshop we demonstrate how to integrate the k-means clustering into the initialization of the execution phase and how ambulance teams can use the results of a trained classifier to decide which victim has a chance of surviving the rescue operation.

Currently, teams need to implement their own custom algorithms to solve rescue tasks including the development of standard artificial intelligence algorithms from scratch, such as k-means clustering or shortest-path planning. MATLAB®, however, provides more efficient, diverse and robust implementations of these standard algorithms that teams may benefit of in order to prioritize other aspects or challenges in the rescue scenario.

Although efficient and robust in isolation, the time constraint imposed on the rescue agents demand a more elaborate assessment of the efficiency of the

[7] `http://www.robocupathomeedu.org/learn`

[8] `https://blogs.mathworks.com/deep-learning/2017/10/06/deep-learning-with-matlab-r2017b/`

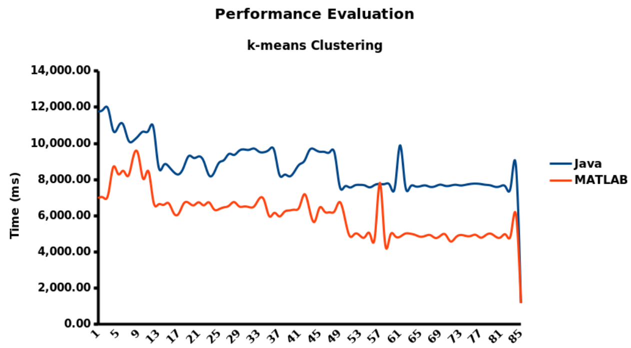

Fig. 7. Performance of the k-means implementation in Java and in MATLAB®.

MATLAB® algorithms integrated to the ADF. We have assessed the performance of the k-means clustering algorithm implemented in the Sample ADF using pure Java and in MATLAB®. Figure 7 shows that the MATLAB® k-means clustering algorithm executes in less time than the Java implementation. There was significant difference on the execution average time for the MATLAB® ($6,095.87 \pm 1,178.51$ ms) and the Java ($8,783.36 \pm 1,188.22$ ms) implementations; $t(164) = 14.63$, $p < 0.05$. Hence, we can conclude that using MATLAB® k-means clustering reduces the effort and maintenance, and increases the performance of the agent teams.

The integration of MATLAB® models into the ADF is based on the MATLAB® API for Java®[9], which enables Java programs via `MatlabEngine` class to interact with MATLAB® synchronously (`startMatlab` method) or asynchronously (`startMatlabAsync` method). In addition to start MATLAB®, there is also a possibility to connect synchronously (`connectMatlab` method) or asynchronously (`connectMatlabAsync` method) to an existing shared instance. To share a MATLAB® instance, enter the command `matlab.engine.shareEngine` in the MATLAB® command window. Once connected, then it is possible to evaluate a MATLAB® function with arguments (`feval` and `fevalAsync` member functions) or evaluate a MATLAB® expression as a string (`eval` and `evalAsync` member functions). Additionally, it is possible to interact with the MATLAB® workspace by getting (`getVariable` and `getVariableAsync` member functions) or setting (`setVariable` and `setVariableAsync` member functions) variables. Once finished the interaction with MATLAB®, disconnect from the current MATLAB® session using `disconnect`, `quit`, or `close` member functions.

The k-means clustering can be integrated into the ADF and executed in the precompute phase or execution phase. In the workshop, we show how to integrate the k-means clustering algorithm in the agents' initialization stage of the execution phase.

The pseudo-code of such integration is shown below.

[9] `https://www.mathworks.com/help/matlab/matlab-engine-api-for-java.html`

```
// Prepare data for Matlab k-means clustering
double[][] mlInput = new double[this.entities.size()][2];
for ( StandardEntity entity : this.entities ) {
  Pair<Integer, Integer> location = this.worldInfo.getLocation( entity );
  mlInput[i][0] = location.first();
  mlInput[i][1] = location.second();
  i++;
}

// Run k-means clustering
Object[] mlOutput = ml.feval( 2, "kmeans", (Object) mlInput, this.clusterSize,
                              DISTANCE, this.distanceMetric,
                              MAX_ITER, this.maxIter );
double[] mlIndex = (double[]) mlOutput[0];
double[][] mlCenter = (double[][]) mlOutput[1];
```

Once connected to MATLAB®, the data needs to be prepared for clustering (i.e., the x and y entity location). Next, the MATLAB® function `kmeans` is evaluated using the `feval` method function with several parameters: the dimension of the k-means output (set: `2`), the number of clusters (default: `10`), the distance metric (default: `cityblock`), and the maximum number of interaction (default: `100`). Once executed, the `feval` returns an object array with the indices (i.e., position 0) and the centers (i.e., position 1) that are casted to their respective data types. Finally, the engine is closed and the indices and the centers can be used to assign agents to specific partitions of the map.

This integration of the clustering algorithm requires only a single call per agent to the MATLAB® engine as its results are stored and used in the reminder of the simulation run. Path planning algorithms, however, are affected by changes in the environment and they may require reprocessing to account for these changes. Because of this characteristic, the MATLAB® path planning algorithm is called every time the agent has to calculate a path during the execution phase of the simulation run, even though the first execution can be performed during the precompute phase. Please see Section 2.3 to further details about the path planning.

Ambulance teams can also benefit of MATLAB® to optimize their rescue operations by predicting more accurately the chance of potential victims to survive. First, however, it is necessary to train a classifier with data collected from earlier runs. This training is performed using the Classification Learner MATLAB® app described in Section 2.2.

We have trained a classifier using the data from rescued victims collected from several simulation executions of the Paris map using a simple rescue team. The data collected was the time (`sTime`), the distance to the nearest refuge (`sDist`), the HP of the victim (`sHP`) and the damage (`sDamage`) at the start of the rescue operation and the HP of the victims (`eHP`) at the end of the rescue operation. We categorized this data depending on the final HP (`eHP`) according to the ranges: 0 `Dead`, 1 – 3000 `Critical`, 3001 – 7000 `Injured`, and 7001 – 10000 `Stable` using the MATLAB® script.

```
// TData is the training data
hp_bins = [0 1 3000 7000 10000];
bin_names = {'Dead', 'Critical', 'Injured', 'Stable'};
TData.hp_class = discretize(TData.eHP, hp_bins, 'categorical', bin_names);
```

Once finished, the trained classifier code (`targetSelectorModel`) is exported using the `Export Model - Export Compact Model` feature and validated against a separate set of validation data with the MATLAB® script below.

```
// VData is the validation data
predictions = targetSelectorModel.predictFcn(VData);

numCorrect = nnz(predictions == VData.hp_class);
validationAccuracy = numCorrect/size(VData,1);
fprintf('Validation accuracy: %.2f%%\n', validationAccuracy * 100);
```

The training and validation steps compose an iterative process whose cycle should be repeated until the validation accuracy is satisfactory. Once this is achieved, the exported model can be saved as a file (`targetSelectorModel.mat`) for future use and a MATLAB® script function created.

```
function predictions = selectTargets(time,dist,hp,damage)
  persistent targetSelectorModel
  if isempty(targetSelectorModel)
     load targetSelectorModel targetSelectorModel
  end

  predictors = table(time,dist,hp,damage, ...
       'VariableNames',{'sTime','sDist','sHP','sDamage'});

  predictions = int32(targetSelectorModel.predictFcn(predictors));
end
```

This function can then be called inside the `calc` method of the `HumanDetector` class for the ambulance team agents using the following code snippet.

```
// rescueTarget is an object containing victim's information
if ( MatlabEngine.findMatlab().length > 0 ) {
  MatlabEngine ml = MatlabEngine.connectMatlab();
  int sTime = rescueTarget.sTime;
  int sDist = rescueTarget.sDist;
  int sHP = rescueTarget.sHP;
  int sDamage = rescueTarget.sDamage;

  int value = ml.feval( "selectTargets", sTime, sDist, sHP, sDamage );

  ml.close();
}
```

This code executes the MATLAB® script function `selectTargets` using data about a specific victim and returns a prediction about the state of the victim at the end of the rescue operation coded as 0 `Dead`, 1 `Critical`, 2 `Injured`, and 3 `Stable`. The ambulance team can then combine this information with several other information about other victims to determine which victim it is worth rescuing. Possible policies to use this classification includes classify all known victims, discard the predicted dead, and

1. select randomly among them
2. select the closest one
3. select the closest one that is predicted `Critical`

Notice that we use `MatlabEngine.findMatlab()` and `MatlabEngine.connectMatlab()` methods instead of `MatlabEngine.startMatlab()`. This requires that a MATLAB® session is running and shared to the code to work. To share a MATLAB® session, open MATLAB®, enter the command `matlab.engine.shareEngine` in its command window, and leave it open during the execution of the simulation.

4 Conclusion

By giving a workshop to the RoboCup teams on how to separate machine learning algorithms from the actual code to control the agents / robots, teams can concentrate on the learning aspect. The algorithms implemented in the workshop are examples of common challenges in this competition, but the approach should extend to any algorithm available/developed in MATLAB®. This approach can be extended to, for instance, deep learning, state machines, and graph node refining algorithms. This approach can give a boost to the scientific level of the RoboCup Rescue Simulation League as

1. Teams may focus more on the development of more high-level algorithms to solve rescue challenges and
2. MATLAB® will provide a performance benchmark against which teams can show their improvements.

The workshop, available online[10], could also attract more teams to the competition, increasing both the level and impact for this socially significant domain.

References

1. Murphy, R.R., Tadokoro, S., Kleiner, A.: Disaster robotics. In Siciliano, B., Khatib, O., eds.: Springer Handbook of Robotics. Springer (2016) 1577–1604
2. Tadokoro, S., Kitano, H., Takahashi, T., Noda, I., Matsubara, H., Shinjou, A., Koto, T., Takeuchi, I., Takahashi, H., Matsuno, F., Hatayama, M., Nobe, J., Shimada, S.: The RoboCup-Rescue project: A robotic approach to the disaster mitigation problem. In: Proceedings of the IEEE International Conference on Robotics and Automation. (2000)
3. Fassaert, M.L., Post, S.B.M., Visser, A.: The common knowledge model of a team of rescue agents. In: Proc. 1th International Workshop on Synthetic Simulation and Robotics to Mitigate Earthquake Disaster. (July 2003)
4. dos Santos, D.S., Bazzan, A.L.: Distributed clustering for group formation and task allocation in multiagent systems: A swarm intelligence approach. Applied Soft Computing **12**(8) (2012) 2123 – 2131
5. Parker, J., Nunes, E., Godoy, J., Gini, M.: Exploiting spatial locality and heterogeneity of agents for search and rescue teamwork*. Journal of Field Robotics **33**(7) (2016) 877–900
6. Lloyd, S.P.: Least squares quantization in pcm. IEEE Transactions on Information Theory **28** (1982) 129–137
7. Kaufman, L., Rousseeuw, P.J.: Clustering by means of medoids. In Dodge, Y., ed.: Statistical Data Analysis Based on the L_1–Norm and Related Methods. North-Holland (1987) 405–416
8. Kaufman, L., Rousseeuw, P.J.: Divisive analysis (program diana). In: Finding Groups in Data. John Wiley & Sons, Inc. (2008) 253–279
9. Marin, J.M., Mengersen, K., Robert, C.P.: Bayesian modelling and inference on mixtures of distributions. In Dey, D., Rao, C., eds.: Bayesian ThinkingModeling and Computation. Volume 25 of Handbook of Statistics. Elsevier (2005) 459 – 507

[10] `https://github.com/IntelligentRoboticsLab/Joint-Rescue-Forces/wiki`

10. Baum, L.E., Petrie, T.: Statistical inference for probabilistic functions of finite state markov chains. Ann. Math. Statist. **37**(6) (12 1966) 1554–1563
11. Corke, P.: Robotics, Vision and Control: Fundamental Algorithms In MATLAB® Second, Completely Revised. Volume 118 of Springer Tracts in Advanced Robotics. Springer (2017)
12. Sheh, R., Schwertfeger, S., Visser, A.: 16 years of robocup rescue. KI - Künstliche Intelligenz **30**(3) (Oct 2016) 267–277
13. Visser, A., Amigoni, F., Shimizu, M.: The future of robot rescue simulation workshop - an initiative to increase the number of participants in the league. University of Amsterdam, Politecnico di Milano & Chukyo University (January 2016)
14. Takami, S., Takayanagi, K., Jaishy, S., Ito, N., Iwata, K. In: Design of Agent Development Framework for RoboCupRescue Simulation. Springer International Publishing, Cham (2018) 185–199