






Integrating the Latest Artificial Intelligence Algorithms into the RoboCup Rescue Simulation Framework

Arnoud Visser¹ , Luis G. Nardin² , and Sebastian Castro³ 

¹ Universiteit van Amsterdam, Amsterdam, The Netherlands
A.Visser@uva.nl

² Brandenburg University of Technology, Cottbus, Germany
nardin@b-tu.de

³ MathWorks, Natick, MA, USA
Sebastian.Castro@mathworks.com

Abstract. The challenge of the Rescue Simulation League is for a team of robots or agents to learn an optimal response to mitigate the effects of natural disasters. To operate optimally, several problems have to be jointly solved like task allocation, path planning, and coalition formation. Solve these difficult problems can be quite overwhelming for newcomer teams. We created a tutorial that demonstrates how these problems can be tackled using artificial intelligence and machine learning algorithms available in the MATLAB[®] and the Statistics and Machine Learning Toolbox[™]. Here we show (1) how to analyze and model disaster scenario data for developing rescue decision-making algorithms, and (2) how to incorporate state-of-the-art machine learning algorithms into Rescue Agent Simulation competition code using the MATLAB[®] Engine API for Java.

Keywords: Machine learning · MATLAB[®] · Rescue Agent Simulation

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 [8], benchmarks based on them, such as the RoboCup Rescue competitions, are ideal for assessing the capabilities of these systems. Thus, one goal of the RoboCup Rescue competitions is to compare the performance of algorithms that coordinate and control teams of either robots or agents performing disaster mitigation tasks.

In particular, the Rescue Agent Simulation competition aims to simulate large scale natural disasters, such as earthquakes, enabling the exploration of new forms of autonomous coordination of heterogeneous rescue teams under adverse conditions. This competition was first demonstrated in the RoboCup

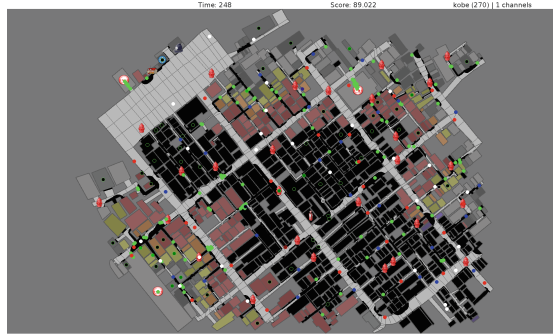


Fig. 1. View of disaster scenario in the Kobe map after an earthquake.

2000 [12] and officially launched in the RoboCup 2001. Participating teams have their background mainly from artificial intelligence and robotics.

The competition is based on a simulation platform and a set of complex scenarios representing the conditions of cities after an earthquake (see Fig. 1). In each scenario, fire brigade, police force and ambulance team agents extinguish fires, unblock roads, and rescue civilians trapped inside collapsed buildings, respectively. The final score in the scenario is calculated based on the number of rescued civilians and the number of remaining buildings taking into account the damage caused by the fire. Scenarios typically contain up to 5000 buildings and up to 1000 civilians, as well as agent teams of fire brigades, police forces and ambulance teams composed of up to 50 agents each.

The complexity of these scenarios 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 [1]. Artificial intelligence (AI), in particular machine learning (ML) algorithms are very well suited to cope with some of these challenges. For instance, fire brigades and ambulance teams can optimize their task allocation decisions by estimating, respectively, the danger of fire ignition in different buildings (discrete state—classification) and the chance of rescuing trapped civilians alive (continuous state—regression).

The implementation of state-of-the-art AI and ML algorithms, their training, and their integration into the Rescue Agent Simulation competition code can be quite overwhelming for newcomer teams. Hence we propose that competition teams take advantage of existing and well-established AI and ML tools to develop their competition code. Here, we demonstrate¹

1. how to use MATLAB[®] and add-on packages, such as the Statistics and Machine Learning Toolbox[™], to analyze and model disas-

¹ All data as well as Java and MATLAB[®] code used to generate the results presented in this work are available at <https://github.com/IntelligentRoboticsLab/Joint-Rescue-Forces> repository.

ter scenario data using both interactive design tools (GUIs) and programming code. The analysis and modeling provide support to the development of more elaborate data-driven rescue decision-making algorithms (see Sect. 2).

2. how state-of-the-art ML algorithms can be directly incorporated into the Agent Development Framework (ADF) [13] using the MATLAB[®] Engine API for Java (see Sect. 3).

2 Interactive Approach

MATLAB[®] and the Statistics and Machine Learning Toolbox[™] can be used in an interactive mode to analyze disaster scenario data and create models that agents can use to base their decisions during the unfolding of these disaster scenarios.

2.1 Unsupervised Methods

Unsupervised machine learning methods can be used to analyze and model disaster scenario data. In the Rescue Agent Simulation competition, clustering algorithms are interesting for agents to partition maps into sectors and evenly distribute the search and rescue workload among them [9, 10]. MATLAB[®] implements several clustering algorithms, such as k-means [6], k-medoids [4], hierarchical clustering [5], Gaussian mixture models [7], and hidden Markov models [2].

We can, for instance, use the MATLAB[®] interactive mode to assess which of these clustering algorithms provides a more evenly distributed number of buildings per sector for a specific city map. This assessment first requires that all (x, y) coordinates of buildings in the city map to be exported into a text file, which can be accomplished including some Java code into precompute phase of the agents code (see `AbstractSimpleAgent.java` lines 79–106). Next, these coordinates are imported into a matrix in MATLAB[®] using the `textscan` command and subsequently partitioned using one of the clustering algorithms available.

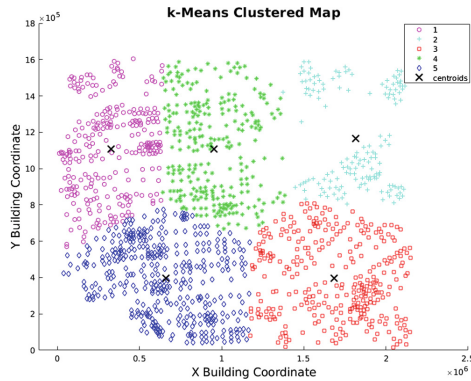


Fig. 2. Partitioning of the buildings in the Paris map using MATLAB[®] k-means clustering algorithm. (Color figure online)

Figure 2 shows the buildings from the Paris map partitioned using the k-means clustering algorithm. The different shapes and colors represent the association of each building to a specific sector. This partitioning was created with the `[indices,centroids] = kmeans([x,y], 5)` command, and the plot generated with the `gscatter(x,y,indices)` command (see `importBuildingsData.m` and `building_cluster.m` scripts).

2.2 Supervised Methods

Supervised machine learning methods can be used to learn associations between variables (part of a causal model of the world). Estimates of variables' value of a world model can be done with discrete states (classification) or continuous states (regression). In the Rescue Agent Simulation, competition teams can use these methods to assess their strategy and the most relevant predictors, for instance, for ambulance teams to estimate the chance that trapped civilians have to survive to a rescue operation by predicting their remaining health points (HP) at the end of a scenario simulation.

To demonstrate the use of MATLAB[®] to evaluate the strategy of a simple agent team, we collected several metrics from multiple runs in multiple scenarios of this agent team and assessed them using available classification and regression supervised learning algorithms in MATLAB[®]. The metrics collected were (see `matlab.generator.simple.agent.ambulance.SimpleAmbulanceTeam.java` lines 389–420): start and end time of the rescue operation (`sTime` and `eTime`), initial and final Euclidean distance to the nearest refuge (`sDist` and `eDist`), initial and final HP (`sHP` and `eHP`), initial and final damage level (`sDamage` and `eDamage`), and initial buriedness (`sBuriedness`).

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. We evaluated the accuracy of different combinations of predictor metrics and concluded that only the metrics with values of the beginning of the rescue operation were relevant to the prediction accuracy (i.e., `sTime`, `sDist`, `sHP`, and `sDamage`).

To use classification, we discretized the `eHP` according to the ranges: 0 **Dead**, 1–3000 **Critical**, 3001–7000 **Injured**, and 7001–10000 **Stable**. Then we trained different classification algorithms in MATLAB[®] using this data and the most accurate classification was obtained using the Weighted K-nearest neighbors (KNN).

Figure 3 shows the Weighted 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 a scenario simulation. This classification predicts correctly 78.9% of the civilians' state. Notice, however, that most of the wrong detections (i.e., sum of the numbers in the red cells) are above the diagonal green cells in the right panel of Fig. 3 meaning that this trained classifier predicts a civilian in a less severe state than

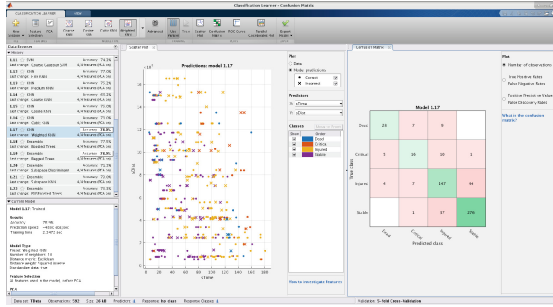


Fig. 3. Classification Learner MATLAB[®] app showing predictions of the injury class of the civilians at the end of the scenario using Weighted K-Nearest Neighbors. **Left panel** shows different assessed classifying algorithms and their respective accuracy. **Middle panel** shows a scatter plot showing the relationship between the initial distance to refuge (sDist) and the time the rescue initiated (sTime) with the model predictions and their correctness. **Right panel** shows the number of predicted versus true (or correct) classification of rescue civilians. The diagonal (green) shows the number of correct classifications, while all other cells represent the number of misclassifications, how they were classified versus the correct classification. (Color figure online)

the civilian really will be. For instance, there are 9 cases in which the civilian will die and the classifier predicted it as injured.

We applied regression methods to the same data without discretizing the eHP and trained different regression algorithms in MATLAB[®]. Figure 4 shows 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 HP values equals 1167.5 (and normalized RMSE equals 0.1228).

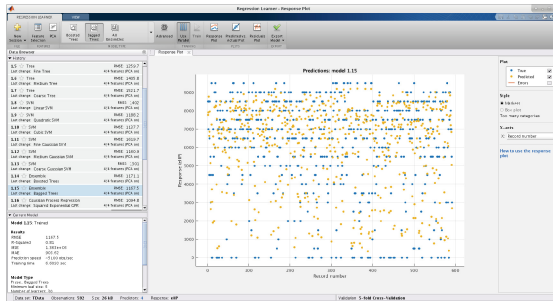


Fig. 4. Regression Learner MATLAB[®] app showing predictions of the chance to survive (remaining HP) of trapped civilians. **Left panel** shows different assessed regression algorithms and their respective root mean square error. **Right panel** shows the prediction versus true HP value of the trapped civilians at the end of a simulation.

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 path planning algorithms from the MATLAB[®] graph and network algorithms² or (2) to use graph-routines from Peter Corke's Robotics Toolbox [3].

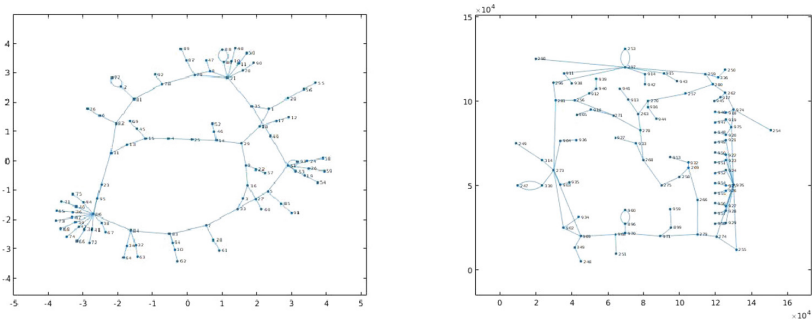


Fig. 5. The small Test map of the RoboCup Rescue Agent Simulation competition in MATLAB[®] as topological (left) and metrical graph (right).

First, however, all the roads of a city map need 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 is called during the precompute phase of the Rescue Agent Simulation simulation, 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 : areaNeighbours) {
        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)`, that loads the graph `G`, calls the MATLAB[®] method `[TR,D]=shortestpathtree` 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*`, which is available in Peter Corke's robotics toolbox [3]. The MATLAB[®]

² <https://www.mathworks.com/help/matlab/graph-and-network-algorithms.html>.

code of this algorithm is open source and well documented making 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 s (measurement with a computer with a Intel Core i7-8550U processor), for larger maps like Paris 22 s are needed for this conversion (see Fig. 6). Both are fast enough for the precompute phase of the competition.

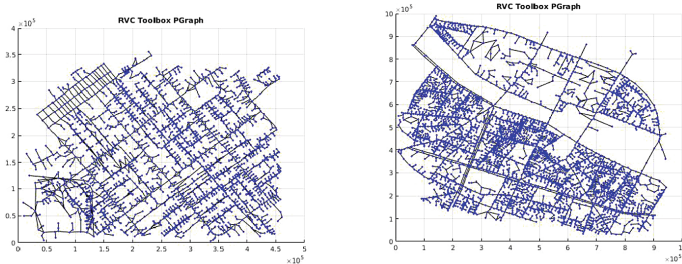


Fig. 6. Maps of Kobe (left) and Paris (right) in Peter Corke's `Pgraph`-format.

An advantage of this approach compared to the path-planning methods typically applied by the Rescue Agent Simulation competition 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

A challenge in the Virtual Robot competition is that whenever an agent reaches a building, it has to enter that building [11]. The MATLAB[®] Robotics System Toolbox allows to directly control robots and realistic simulation via the Robotics Operating System (ROS) interface, as demonstrated in the Future of RoboCup Rescue workshop [14] and the RoboCup@Home Education workshop³.

Another challenge in this competition is the detection of buried victims from camera images. In the same workshops, victims detection has been demonstrated using the MATLAB[®] deep learning capabilities, a combination of the Neural Network Toolbox, Parallel Computing Toolbox, GPU Coder, and Computer Vision System Toolbox. Notice that these toolboxes run models deployed to GPU faster than TensorFlow or Caffe, which is a highly desirable for robotic applications⁴.

³ <http://www.robocupathomeedu.org/learn>.

⁴ <https://blogs.mathworks.com/deep-learning/2017/10/06/deep-learning-with-matlab-r2017b/>.

3 ADF Integration

In addition to using the MATLAB[®] models and algorithms in interactive mode, they can also be integrated into the Agent Development Framework (ADF) [13] to run during the simulation execution. ADF is the mandatory agent architecture for all competition teams participating 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.

The integration of MATLAB[®] models into the ADF is based on the MATLAB[®] Engine API for Java[®]⁵, 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, it is possible to evaluate a MATLAB[®] function with arguments (`feval` and `fevalAsync` functions) or evaluate a MATLAB[®] expression as a string (`eval` and `evalAsync` functions). Additionally, it is possible to interact with the MATLAB[®] workspace by getting (`getVariable` and `getVariableAsync` functions) or setting (`setVariable` and `setVariableAsync` functions) variables. Once finished the interaction, disconnect from the current session using `disconnect`, `quit`, or `close` functions.

In Sect. 3.1 we show how to integrate the k-means clustering into rescue agents, and in Sect. 3.2 how ambulance team agents can use a trained classifier to decide which trapped civilian has a better chance of surviving a rescue operation.

3.1 Clustering Integration

Currently, competition teams need to implement their own version of standard artificial intelligence algorithms from scratch to solve common tasks, such as k-means clustering. However, MATLAB[®] provides more diverse and robust implementations of these standard algorithms that teams may benefit of to prioritize the development of high-level strategies.

Although diverse and robust, the time constraint imposed on rescue agents demands a more elaborate assessment of the efficiency of the MATLAB[®] algorithms integrated to the ADF. Here, we have assessed the performance of the k-means clustering algorithm implemented in the Sample ADF using pure Java and in MATLAB[®] measured in a computer with Intel Core i7 6700HQ 2.6 GHz (8 cores) processor and 16 GB RAM using Arch Linux, Oracle Java JDK 8 and MATLAB[®] R2017b.

Figure 7 shows the result of this assessment in which the MATLAB[®] k-means clustering algorithm executes in less time than the Sample ADF Java

⁵ <https://www.mathworks.com/help/matlab/matlab-engine-api-for-java.html>.

implementation for all 83 agents. 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.

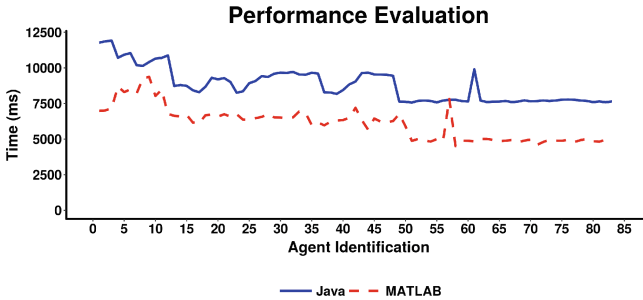


Fig. 7. Performance of the k-means implementation in Java and in MATLAB[®] performed in sequence by 83 agents during the initialization stage of the execution phase of the scenario simulation for the Kobe map.

The k-means clustering can be integrated into the ADF and executed in the precompute phase or execution phase of the scenario simulation. Here, we show how to integrate the k-means clustering algorithm in the agents' initialization stage of the execution phase. The Java code for such integration is

```
// 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];
```

This code connects the agents to MATLAB[®] and prepares entities data for clustering (i.e., the x and y entity location). In the initialization, agents are executed in sequence avoiding concurrent MATLAB[®] connections. Next, the MATLAB[®] function `kmeans` is evaluated using the `feval` method with several parameters: the dimension of the k-means output (set to 2), the number of clusters (set to 10), the distance metric (set to `cityblock`), and the maximum number of interaction (set to 100). Once executed, the `feval` method returns an object array with the indices in the position 0 and the centers in the position 1 that are cast 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.

The 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 remainder of the simulation run. Path planning algorithms, for example, are affected by changes in the environment, and would require reprocessing to account for changes. Because of this characteristic, the MATLAB[®] path planning algorithm would need to be called every time the agent has to calculate a path in the execution phase of the simulation run, even though the first execution can be performed during the precompute phase. Please see Sect. 2.3 to further details about the path planning.

3.2 Classifier Integration

Ambulance teams can also benefit of MATLAB[®] to optimize their rescue operations by predicting more accurately the chance of trapped civilians to survive a rescue operation. 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 as described in Sect. 2.2.

We have trained a classifier using the data from rescued civilians 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 civilian HP (`sHP`) and the damage (`sDamage`) at the start of the rescue operation and the HP of the civilian (`eHP`) at the end of the rescue operation. We discretized the final HP (`eHP`) according to the ranges: 0 `Dead`, 1–3000 `Critical`, 3001–7000 `Injured`, and 7001–10000 `Stable` using the code

```
// 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);
```

The trained classifier model (`targetSelectorModel`) is exported using the `Export Model - Export Compact Model` feature and validated against a validation dataset with the code

```
// 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 comprise an iterative process whose cycle should be repeated until the validation accuracy is satisfactory. Then, the exported model can be saved as a file (`targetSelectorModel.mat`) and invoked in the function

```
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

```

The `predictions` function can then be called inside the `calc` method of the `HumanDetector` class for the ambulance team agents in the ADF using the code

```

// 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[®] 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 prediction with several other information about other victims to determine which victim is worth rescuing first. Possible strategies to use this classification includes (1) classify all known victims, (2) discard the predicted dead, and (3)

- a. select one randomly among them
- b. select the closest one
- c. select the closest one that is predicted **Critical**

Notice that we use `MatlabEngine.findMatlab()` and `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

This paper describes the possible uses of existing artificial intelligence (AI) and machine learning (ML) tools to analyze and model disaster scenario data as well as the integration of these tools to the competition code. The examples provided tackle common challenges of the Rescue Agent Simulation competition in which AI and ML tools suit. The approach, however, is extensible to any other algorithm available in MATLAB[®] or any other tool that provides an interface in Java. For instance, this approach can be extended to integrate deep learning, state machines, and graph node refining algorithms, which may increase the scientific outcomes of the Rescue Simulation League as

- (1) teams may focus on high-level strategies to solve rescue challenges and
- (2) MATLAB[®] will provide a performance benchmark against which teams can show their improvements.

References

1. Akin, H.L., Ito, N., Jacoff, A., Kleiner, A., Pellenz, J., Visser, A.: RoboCup rescue robot and simulation leagues. *AI Mag.* **34**(1), 78–87 (2013). <https://doi.org/10.1609/aimag.v34i1.2458>
2. Baum, L.E., Petrie, T.: Statistical inference for probabilistic functions of finite state Markov chains. *Ann. Math. Stat.* **37**(6), 1554–1563 (1966). <https://doi.org/10.1214/aoms/1177699147>
3. Corke, P.: *Robotics, Vision and Control: Fundamental Algorithms In MATLAB[®] Second, Completely Revised*. Springer Tracts in Advanced Robotics, vol. 118. Springer, Heidelberg (2017). <https://doi.org/10.1007/978-3-319-54413-7>
4. 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*, pp. 405–416. North-Holland (1987)
5. Kaufman, L., Rousseeuw, P.J.: Divisive analysis (program DIANA). In: *Finding Groups in Data*, pp. 253–279. Wiley (2008). <https://doi.org/10.1002/9780470316801.ch6>
6. Lloyd, S.P.: Least squares quantization in PCM. *IEEE Trans. Inf. Theory* **28**, 129–137 (1982). <https://doi.org/10.1109/TIT.1982.1056489>
7. Marin, J.M., Mengersen, K., Robert, C.P.: Bayesian modelling and inference on mixtures of distributions. In: Dey, D., Rao, C. (eds.) *Bayesian Thinking Modeling and Computation, Handbook of Statistics*, vol. 25, pp. 459–507. Elsevier (2005). [https://doi.org/10.1016/S0169-7161\(05\)25016-2](https://doi.org/10.1016/S0169-7161(05)25016-2)
8. Murphy, R.R., Tadokoro, S., Kleiner, A.: Disaster robotics. In: Siciliano, B., Khatib, O. (eds.) *Springer Handbook of Robotics*, pp. 1577–1604. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-32552-1_60
9. Parker, J., Nunes, E., Godoy, J., Gini, M.: Exploiting spatial locality and heterogeneity of agents for search and rescue teamwork. *J. Field Robot.* **33**(7), 877–900 (2016). <https://doi.org/10.1002/rob.21601>
10. dos Santos, D.S., Bazzan, A.L.: Distributed clustering for group formation and task allocation in multiagent systems: a swarm intelligence approach. *Appl. Soft Comput.* **12**(8), 2123–2131 (2012). <https://doi.org/10.1016/j.asoc.2012.03.016>
11. Sheh, R., Schwertfeger, S., Visser, A.: 16 years of robocup rescue. *KI - Künstliche Intelligenz* **30**(3), 267–277 (2016). <https://doi.org/10.1007/s13218-016-0444-x>
12. Tadokoro, S., et al.: The RoboCup-rescue project: a robotic approach to the disaster mitigation problem. In: *Proceedings of the IEEE International Conference on Robotics and Automation* (2000). <https://doi.org/10.1109/ROBOT.2000.845369>
13. Takami, S., Takayanagi, K., Jaishy, S., Ito, N., Iwata, K.: Design of agent development framework for RoboCupRescue simulation. In: Lee, R. (ed.) *CSII 2017. SCI*, vol. 726, pp. 185–199. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-63618-4_14
14. 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