# Team RCMakers's Technical Report for the DJI RoboMaster AI Challenge at ICRA 2018

Ali Çataltepe[*], Ege Çağlar[†], Tan Gemicioğlu[‡]
Robert College, Istanbul, Turkey
[*]ali@cataltepe.com, [†]ege1358@gmail.com, [‡]tan@gemdata.com.tr

Contents

## I. Hardware

### A. Mechanical Description

We had to modify the internal middle cage of the Robomaster AI Robot in order to fit the Jetson TX1, the computer that we chose to use as the robot's controller, and the Scanse Sweep LIDAR. The platform carrying the shooter was also raised for this purpose but its height was still within the constraints stated in the rules manual. No outside components were added to the cage or the robot except the custom-built platforms for both Jetson TX1 and Scanse Sweep. An external USB web camera, Microsoft Lifecam HD-3000 was placed on top of the turret mechanism for visual recognition and targeting purposes. A powered USB hub was also installed to allow the Scanse Sweep and Lifecam HD-3000 to be simultaneously connected to the Jetson TX1.

### B. Sensors

In order to be able to detect obstacles and other robots in every direction, a LIDAR was required on the robot. The LIDAR sensor was mainly used in the localization module of the robot. The localization module used the sensor to find the position of the robot relative to the obstacles in the map. The distance to the perceived obstacles (or the distance to the Protection Fence if none are detected) in every cardinal direction was also fed to the LSTM network in the decision-making module. We also needed a camera to make sure we detected other robots, better predict their position and for finding and targeting enemy robots' armor modules.

Due to its range and affordable price, the Scanse Sweep v1.0 LIDAR sensor was used as the 2D LIDAR in the robot. The sensor's horizontal field of view is 360 degrees, while its vertical field of view is 0.5 degrees. It has a range of 40m with a 75% reflective target, has a maximum sample rate of 1075 Hz and has a rotation frequency of 1-10

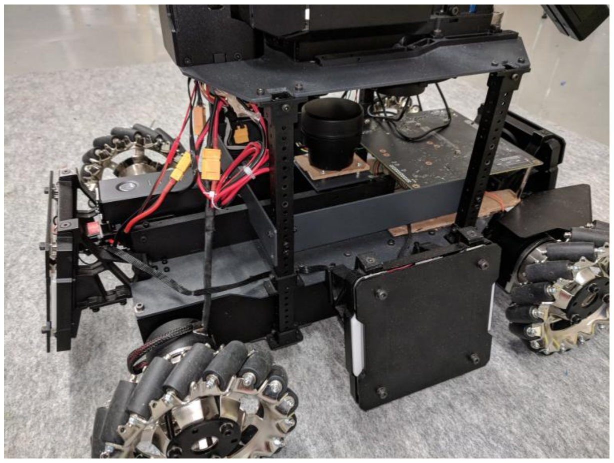


Fig. 1. The middle section of the robot with plates removed to make added components visible

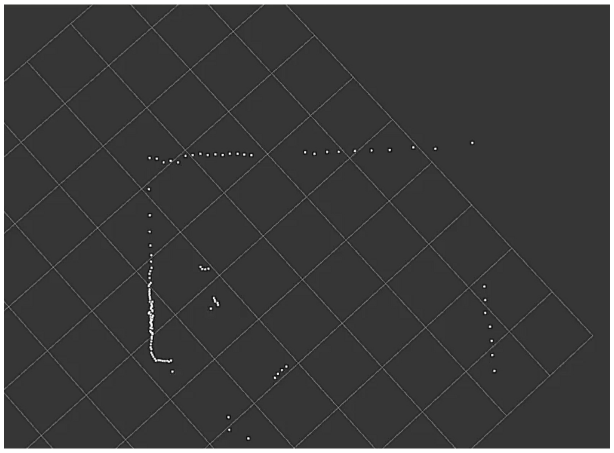


Fig. 2. LIDAR sensor measurements

Hz. It is connected to the NVIDIA Jetson TX1 by the USB hub and it is used to provide a sensor_msgs_/LaserScan message to the localization module as seen in Figure 1. The Scanse Sweep normally has a UART output but it's connected using a USB-to-serial converter to a powered hub as the Jetson TX1's UART and USB ports cannot supply enough power for the Sweep.

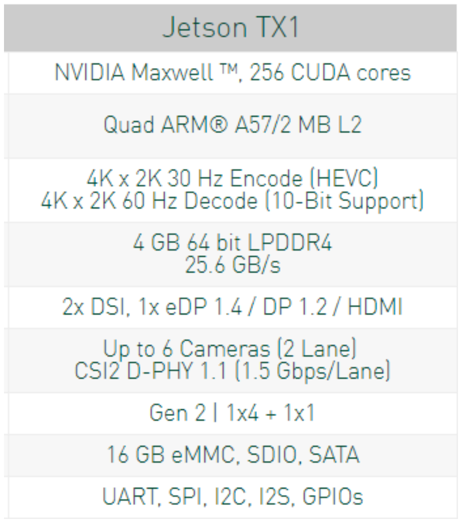
The Microsoft Lifecam HD-3000 Webcam was used as the camera of the robot. The camera has a resolution of 1280 X 720 pixels for videos, has a maximum frame rate of 30 FPS, has 68.5 degrees of diagonal field of view and has fixed focus from 0.3m to 1.5m. The camera is connected to the NVIDIA Jetson TX1 by the USB hub and used with the OpenCV library in the detection module.

### C. Computers

Aside from our personal computers, there are two main computers we have currently used for the challenge. On our robot, we use a Jetson TX1 Development Kit by NVIDIA. Its specifications can be found in Figure 3.

For training, we used a custom-built computer running Ubuntu. It has a Intel i7-4790K CPU and 16GB of 2400MHz DDR3 RAM. The most important component for the simulation is the GPU, a NVIDIA GTX 960. Running our LSTM network with Tensorflow allowed our training to be accelerated by NVIDIA's CUDA toolkit.

During the final week before ICRA 2018, we plan to finalize the parameters of our system and use a NC6s_v3 Linux Virtual Machine by Microsoft Azure to do a extremely accelerated training for one



| Jetson TX1 |
|---|
| NVIDIA Maxwell ™, 256 CUDA cores |
| Quad ARM® A57/2 MB L2 |
| 4K x 2K 30 Hz Encode (HEVC)<br>4K x 2K 60 Hz Decode (10-Bit Support) |
| 4 GB 64 bit LPDDR4<br>25.6 GB/s |
| 2x DSI, 1x eDP 1.4 / DP 1.2 / HDMI |
| Up to 6 Cameras (2 Lane)<br>CSI2 D-PHY 1.1 (1.5 Gbps/Lane) |
| Gen 2 \| 1x4 + 1x1 |
| 16 GB eMMC, SDIO, SATA |
| UART, SPI, I2C, I2S, GPIOs |

Fig. 3. Specifications for the Jetson TX1

day. It has 6 cores of Azure vCPUs, 112 GiB of RAM and a V100 NVIDIA Tesla GPU optimized for high-performance AIs.

## II. Software

### A. Detection

1) Algorithm: The detection module uses the camera and LIDAR module to detect other robots, their locations, whether they are an ally or an enemy, the highest to-hit probability of their armor plates and the aiming angle to the armor plate with the highest to-hit probability. To be able to provide this data, the detection module uses the camera module and detects the LED strips next to the armor plates.

Using the OpenCV library, the module first desaturizes the video feed and uses a simple threshold to identify any bright objects in front of the camera. The video feed is also fed to two different RGB filters, one detecting red and one blue in order to detect the LED strips. Results from both RGB filters are combined separately with the result from the threshold filter to get two separate processed video feeds, one highlighting the red lights and one highlighting the blue lights. The module then detects the strips of light as rectangles and only considers the ones with a height more than its width in order to eliminate the lights on top of the Referee Module. The module then takes the centers of these regions and matches them to the closest neighbor center, creating a model for the armor plates. Since the actual proportions of the armor plates are known,

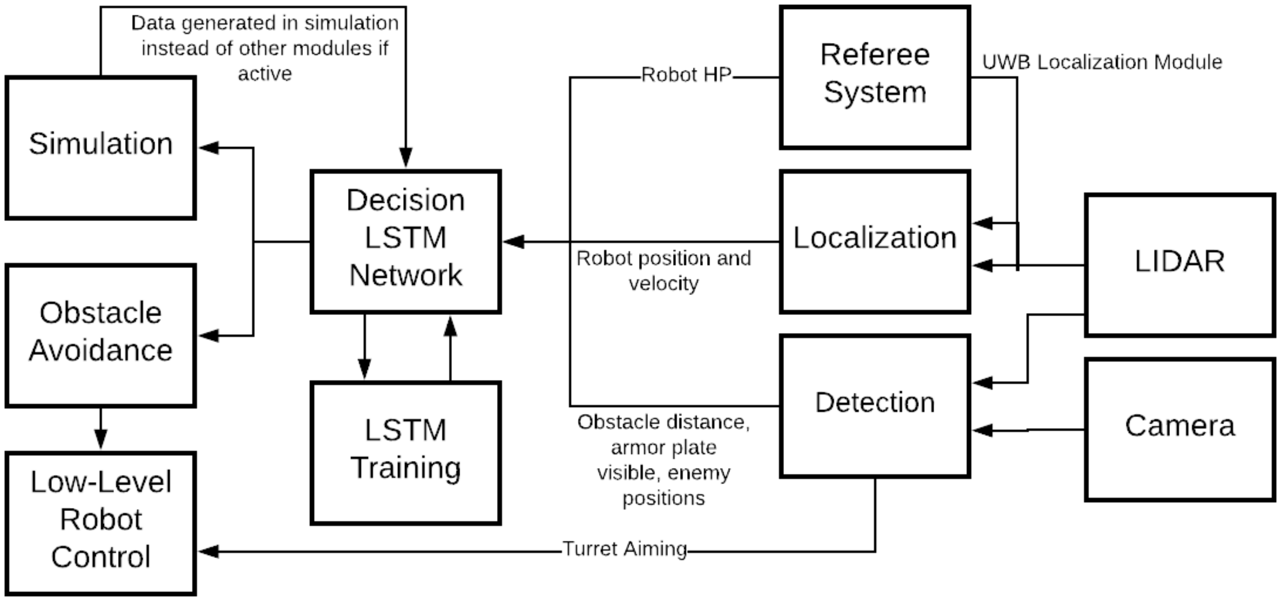


Fig. 4. Diagram outlining the software structure

the distance and the area of the plate is calculated for every plate.

The detection module then uses the LIDAR data and the angle of the camera to find the distance to the robot. Using this knowledge, the detection module finds the location for each robot seen and publishes this information to the decision-making module. The module then selects the enemy robot which has the armor plate with the highest area and calculates its to-hit probability (based on the area of the selected armor plate) and its angle depending on the camera, publishing the information to the decision-making module and the control module.

2) Performance: The detection module has a maximum frequency of 20 Hz, which is sufficient for the robot since the decision-making module can only operate in 10 Hz. The detection algorithm can detect armor plates up to 160 cm for stationary targets. To increase the accuracy of the algorithm, we are currently trying to train a YOLO CNN model as described in the technical proposal, since the accuracy can fall especially when the camera angle is diagonal to the robot and especially because of the red or blue light given by the RFID module under the robot.

## B. Localization

The localization module uses the UWB Location Module and a LIDAR sensor to estimate the position of the robot. The module first takes data from the Scanse Sweep LIDAR modules and converts it to a sensor_msgs/LaserScan message. Since the LIDAR sensor is located inside the robot, any parts surrounding the LIDAR can lead to false measurements, therefore, we remove them before converting the data to a LaserScan message. The message is then used by an Adaptive Monte Carlo Localization algorithm with the static map seen in the RoboRTS package to get a geometry_msgs/PoseWithCovarianceStamped message. This message is then combined with the UWB Location Module and its approx. 10 cm uncertainty and normalized to get the pose estimate and uncertainty of the robot. Using this method, we achieved a result with less uncertainty than both the uncertainty in the UWB Location Module and the LIDAR sensor.

Since we didn't have the resources to make a to-scale practice field, we couldn't test the accuracy of the robot in real life. However, we tested the AMCL node in Gazebo by simulating LIDAR sensors. In the simulation, the pose converged after 20-40 seconds in the beginning of the match and had an

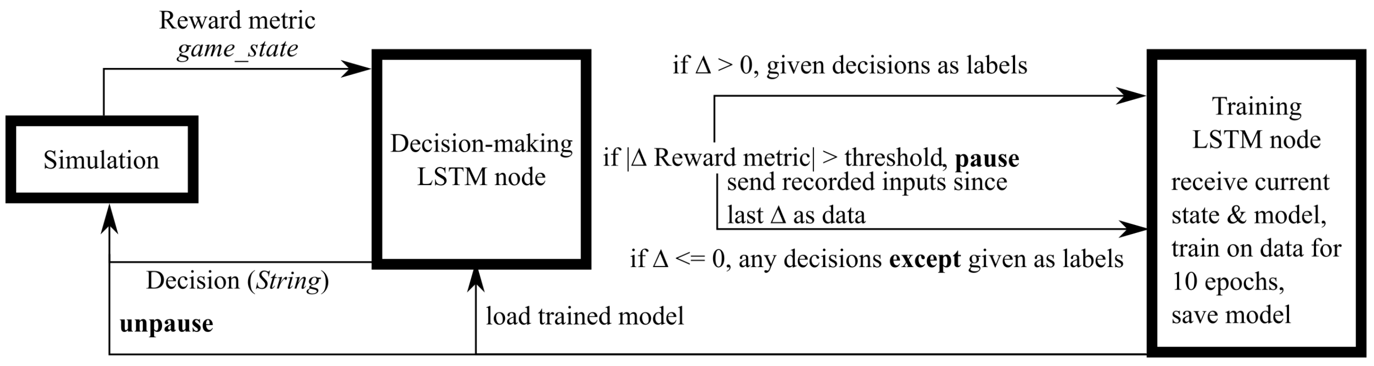

Reward metric
*game_state*
Simulation
Decision-making LSTM node
if Δ > 0, given decisions as labels
if |Δ Reward metric| > threshold, **pause**
send recorded inputs since last Δ as data
if Δ <= 0, any decisions **except** given as labels
Training LSTM node
receive current state & model, train on data for 10 epochs, save model
Decision (*String*)
**unpause**
load trained model

Fig. 5. Diagram showing the training process

approximate uncertainty of 40 mm.

Additionally, because of the localization module's uncertainty, we added an obstacle avoidance node that will block any command from the Decision-making Module to go in a direction if the LIDAR measures an obstacle in that direction closer than 40 mm to the robot.

## C. Decision-making

We found that there were two main paths we could take for what we wanted our learning system to output: High-level macroactions such as travelling to a position and taking aim, or low-level atomic decisions such as moving forward or shooting. As we planned to use a RNN for our main system structure, predicting and picking the best option out of a set of predetermined actions would be much more easier than trying to generate coordinates. This also eliminated the need for separate local and global planners, as the maneuvers of a properly-trained recurrent neural network will result in the emergent generation of strategies in and of themselves.

As detailed in our Technical Proposal, all decision-making and planning has been delegated to a Long Short-Term Memory network (with a cell size of 150 followed by a fully-connected layer of 20, terminating in a softmax classifier with length equal to the number of maneuvers to choose from) outputting the probabilities with which to weight a random choice between a finite set of mutually exclusive atomic maneuvers at approximately 10 Hz. Due to an inability to secure the computing power necessary to run (in real time) and train a convolutional neural network that would extract features from a top-down RTS view of the arena, we elected to numerically represent the game state for to the robot via the following features in a custom game_state ROS message format:

- The 2D linear and angular velocity of the team robot in a Twist message
- The 2D position and orientation of the team robot in the frame of the arena contained in a Pose message
- The estimated 2D position of the enemy robot if detected or the last visible 2D position of the enemy robot in a Pose message
- The relative area of the enemy armor plate currently targeted by the targeting system in the FOV of the robot's camera
- The distance to the nearest obstacle to the front, back, left, and right of the robot
- Current game time (in units of ticks increasing after every action, making a total of 1800 ticks for a round)
- The current HP of the robot

A reward metric (detailed under "Reward Metric"), which was not fed directly into the neural network but processed by the decision-making node to decide when to train during simulated sessions

The moves from among which the LSTM could choose were as follows. Each move, with the exception of shooting, which would shoot a single ball, were treated as continuous until a different decision was published:

- Move forward
- Move backward
- Strafe left
- Strafe right
- Rotate left
- Rotate right
- Stop
- Shoot

As the shooting maneuver was tied to a pre-existing trigger subroutine and the decisions were published at a maximum frequency of 10 Hz to begin with, it would not be possible to exceed limitations on firing frequency or speed.

The algorithm was implemented in Tensorflow using a dynamic_rnn accepting input batches of length 1 (one game_state message), with the state being manually saved and updated as new data was received to effectively execute a real-time LSTM.

The trajectory generation LSTM was initially trained using using supervised learning based on a limited number of human-vs-human matches. The longer training process for the LSTM was accomplished via a Policy Gradient in AI-vs-AI matches in a simulated arena implemented in Gazebo. The training process during a given match for each AI agent was as follows:

1) Two ROS nodes running two different copies of the model were initialized, one for outputting decisions and the other for training the decision-making copy
2) The decision-making copy saved each game state, here referred to as the nth game state, inputted to it into a list containing a sequence of game states starting from the state in which the game started and ending, in sequence, at the second, third...n-1th and nth states. It also saved each decision actually inputted to the game (via the random decision process weighted by the LSTM's output) as a one-hot vector, effectively creating a series of sequences and labels for a sequence classification task ending at each given game state.
3) If the reward metric when a game state was received was found to have changed by more than a predetermined threshold since the start of the match or the last training session, the game was paused and the LSTM's state at the end of the last training session was saved. If the change was negative, each output one-hot vector had its 1.0 unit changed to zero, with the rest changed to have a probability of 0.14 ( 1.0/7) (training would reward any decision except that which was outputted). If the change was positive, the labels remained unchanged (training would reward the decision that was outputted).
4) The training model was started, and the input sequences and outputs were fed to it as a training dataset for a sequence classification task to be backpropagated on using an RMSProp optimizer for 10 epochs. After this, the model's new variables were saved and loaded to the decision-making node, and the match was unpaused.

The process with a human-controlled agent was the same (input recording and automatic backpropagation on a training model in response to changes in a reward metric), except for the lack of a model to output real-time decisions. The models trained using human players were later initialized as the initial decision-making models in the AI-vs-AI matches, which could be accelerated as far as our hardware allowed.

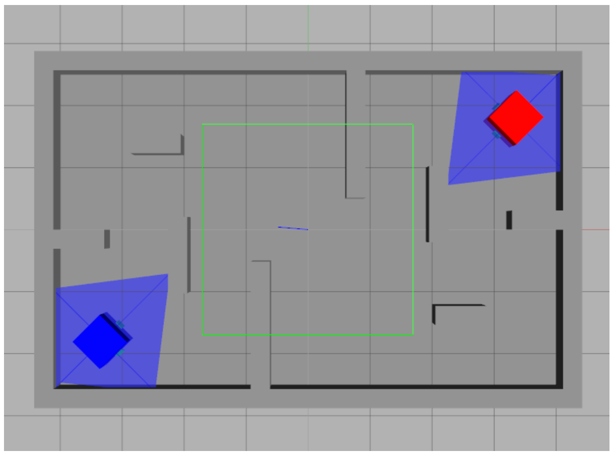


Fig. 6. Top-down view of the simulation

1) Simulation: We wanted our simulation to be as minimalistic as possible for quick training and only created basic models to allow for interaction between the robots. The simulation generates a game_state message and publishes it for the decision-making node. Some of the data, e.g. position and armor plate area are intentionally noisy to simulate real conditions. The simulation also detects the distance to the nearest obstacles using lasers.

2) Reward Metric: Our reward metric (initialized separately for each AI agent) was initialized at 0 at the beginning of each match. Our predetermined change threshold to initiate training was 10. Events which would change the reward metric were as follows:

- Hit taken from an enemy, -5
- Damage inflicted on an enemy, +5

- Entered and stayed in bonus zone for the time required to activate its effects: +10
- Did not change location by more than 2m within 5 seconds, -1
- Collided with an obstacle or robot, -3
- Within 10m of an enemy without taking damage for 3 seconds, +2

The reward metric has gone through a few iterations and we expect there to be more changes before our final version. Currently, we are planning on removing the condition for collisions and forcing the robot to avoid collisions by integrating the obstacle avoidance node into the simulation so that it represents real robot behavior more accurately and doesn't confuse the LSTM network.

3) Performance: Matches were conducted 1v1 (to speed up initial testing), first with players and later 2 separate copies of the model each randomly selecting a version from among their most recent 10 end-of-game iterations of training to prevent overfitting. After 50 human-conducted matches provided a starting point for the models, the models were left to conduct matches among themselves and train on our test machine for 10 hours.

After training was complete, the model with the highest win rate when on its own (52%, 3000 matches total) performed with a 70% win rate (100 matches) when matched against agents controlled by entirely random choices. As we currently lack the resources and time to set up a to-scale practice field (let alone acquire other robots to test our models against), we have not been able to test our decision-making system on the actual robot.

For the next phase of training, the match setup will be changed to 1v2 (as we could only secure a single AI robot due to financial constraints) in order to better represent the actual match. Our game_state message and reward metric will also be modified accordingly. We expect that training will be somewhat slower but different, more successful behavior will emerge on both sides of the match.

## III. Video

Our demonstration video is available on YouTube via the following link: https://youtu.be/UunT6b04FrI

As we only have a single robot and cannot build a practice field, our video is centered around our simulation and more specifically our decision-making system.