ICRA 2019 RoboMaster AI Challenge
Technical Report

1. **Team Name**
   Roboin, Yonsei University (Seoul campus)
2. **Technical Report**
   A. Hardware Description
   i. Mechanical Structure

As we used extra LiDAR, Depth Camera, Jetson TX2, etc., we designed extra supports to carry those parts. The Figure 1,2 shows the CAD drawings of each supports.

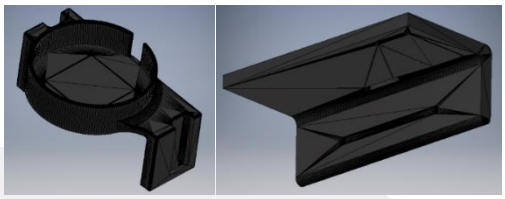

Figure 1 CAD drawing of LiDAR and Depth Camera support

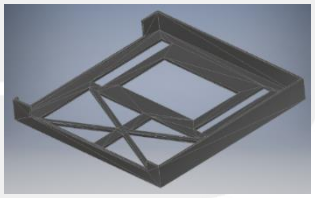

Figure 2 CAD drawing of Jetson TX2 support

ii. Sensor

We used RPLIDAR-A2 for our LiDAR sensor. The Figure 3 shows the object.

Figure 3 RPLIDAR-A2

The Table 1 shows the specifications of LiDAR sensor.

Item	Specification
Distance Range	0.15 ~ 18 m
Angular Range	0 ~ 360°
Distance Resolution	< 1% of actual distance
Angular Resolution	0.45 ~ 1.35°
Sample Duration	0.25 ms
Sample Frequency	2000 ~ 8000 Hz
Scan Rate	5 ~ 15 Hz

Table 1 Specifications of RPLIDAR-A2

As its angular range and distance range is fit for our competition, we use this sensor for locating and navigation. We installed this sensor in front of the robot for accurate measurements.

iii. Computing Device

We used NVIDIA Jetson TX2 for computing device. The Figure 4 shows the

object.

Figure 4 Jetson TX2 Development Kit

The Table 2 shows the specifications of Jetson TX2.

Item	Specification
GPU	NVIDIA Pascal, 256 CUDA cores
CPU	HMP Dual Denver 2/2 MB L2 + Quad ARM A57/2 MB L2
Memory	8GB 128bit LPDDR4 (59.7 GB/s)
Data Storage	32GB eMMC, SDIO, SATA
USB	USB 3.0 + USB 2.0

Table 2 Specifications of Jetson TX2

As its specification is fit for our algorithm, we use this board for our computing device. It was installed on the side of the robot, below the projectile reservoir. To protect it from enemy's attack, we add a guide in front of Jetson board.

iv. Others

We use Intel RealSense Depth Camera D435 for our vision recognition and detection. The Figure 5 shows the object.

Figure 5 Intel RealSense Depth Camera

The Table 3 shows the specifications of Intel RealSense Depth Camera D435.

Item	Specification
Depth FOV for HD 16:9	85.2° × 58° (±3°)
Depth stream output resolution	Up to 1280 × 720
Depth stream output frame rate	Up to 90 fps
Minimum depth distance	0.11 m
Maximum range	Approx. 10 m
RGB sensor resolution & frame rate	1920 × 1080 at 30 fps
RGB sensor FOV	69.4° × 42.5° (±3°)

Table 3 Specifications of Intel RealSense Depth Camera

As its specification is fit for our detecting algorithm, we use this for our camera. It was installed below the gymbol's speed monitor module. So, it is possible to aim on the same line as the direction of gymbol.

B. Software Description

i. Overall Architecture

The overall Architecture resolves into 4 layers: Decision layer, Behavior

layer, Perception layer and Actuator Layer. Figure 6, 7 shows overall architecture.

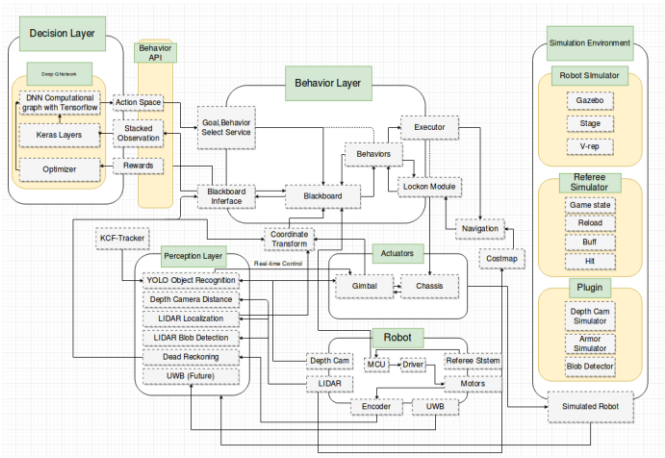

Figure 6 Overall Architecture (1)

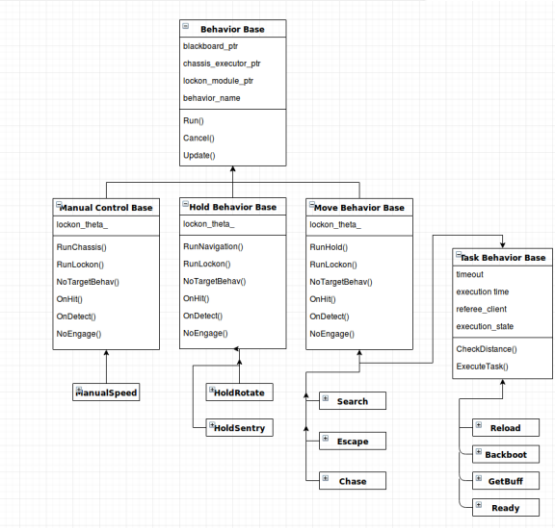

Figure 7 Overall Architecture (2)

Decision layer makes strategy based on perceptions and behavior layer concretizes its execution plan with actuator layer. Agents in perception layer interpret raw sensor data and yield useful information. This information is then sent to Blackboard.

Blackboard serves as a hub for robot information. Every information gathers in blackboard and decision/behavior agents consult that information that are stored in blackboard. Blackboard locates in behavior layer since many behaviors require quick response. Usually the total system spins at 10hz with ROS network. However, Behaviors and executors require different scheduling which makes them incompatible with ROS. Thus, for necessity, blackboard object is instantiated in behavior node.

Decision layer gets processed information from blackboard with API. Only 3 functions are used for this communication: GetInfo, SetBehavior, SetGoal. Behavior and following goal parameter decided by decision layer is concretized in behavior layer. For example, if decision layer requests search behavior to coordinate (4,5), actual plans are made by behaviors. They plan

the path using path planning modules, decides which angle to look at while moving, turn to the direction when hit, whether to lock on enemy, and so on.

This architecture can significantly reduce burden for decision agent. Since every data are processed and interpreted by perception layer and detailed executions are performed by behavior, learnings can be done very easily. If we use deep reinforcement learning as a decision agent, for example, first there is no need to use big convolutional network to process raw camera data. This significantly reduces number of parameters to be optimized.

Second, action space is very small and abstract. The robot doesn't have to start from scratch. Robots with action space of simple x, y velocity need to learn first how to make path to enemy without collision. If rewards are given only when it hits enemy, the reward will be significantly diluted. If one chose to give advantage for other factors, like move distance, this will make model significantly biased. On the other hand, our robot can reach enemy even at its first attempt as it uses navigation goal as action space. This will significantly boost learning.

Lastly, abstract action space helps the robot avoid being biased and overfitted to learning environments. As a result, simulated reinforcement learning parameters are directly applied to real robots. Due to such superiority in architecture, our team succeeded in using simulation trained model in real robot without any modification.

ii. Automatic Recognition and Prediction Algorithm

It is good to recognize an opponent with the current target recognition algorithm, considering the time until the actual order was issued, it is highly likely that the opponent's armor has already moved elsewhere. Currently, we use algorithms that we expect many armors to locate the average location of gimbal, but we're trying to pinpoint armor's position by predicting exactly what the opponent's move.

The strategy algorithm that will be developed in the future is to predict robot movements after 100 ms by looking at human movements and by referring to the paper[1] that predicts the path to move forward and the paper[2] that combines CNN and LSTM to improve accuracy.

So, we decided to apply the LiDAR that we're already using to recognize the direction that the camera isn't looking at. Here's how it works.

Groups of points were grouped together so that groups of a certain size or larger were represented by red lines and groups of below were represented by green circles. So, in this way, we've made robots aware of simple maps that don't have objects smaller than robots. Figure 8 shows how it works.

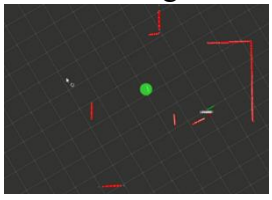

Figure 8 Obstacle detection using 2D LiDAR

iii. Localization

In our algorithm stochastic locating of robot with LIDAR is performed with particle filter and weighted Monte Carlo sampling. Bayesian inference and motion update are performed for each single step in Markov process to stochastically evaluate individual particles. Each particle is weight sampled, thus converges quickly and unbiased. Number of particles are adaptively reduced or increased with respect to its deviations. Other sources besides LIDAR and dead reckoning, such as UWB can be integrated to estimation but not implemented yet in our project. Open source ROS AMCL package is used with parameter adjustment for holonomic motions.

iv. Motion Planning

Motion Planning Consists of Two parts: Global planner and local planner.

1. Global Planner

Our global Planner uses Dijkstra's algorithm in path findings. Although A* algorithm is faster than Dijkstra's, A* doesn't always yields optimal solution depending on which heuristic function used. In our motion planning, not only the distance but the weighted distance from obstacles are also reflected to the heuristic functions. Also, as obstacle avoidance is much more emphasized than distance-optimal path, using A*, which is somewhat lazy in avoiding obstacles doesn't seem to be a good option for us. Thus, our robot planner used Dijkstra's as it is expected to navigate through complex costmaps. Our planner uses ROS navfn package.

2. Local Planner

Local Planner makes velocity plan using our Chassis Lock-on algorithm to exploit the advantage of holonomic motions. Our Chassis Lock-on algorithm supports any transversal trajectory with arbitrary angular velocity, thus enables our 'Lock-on' feature. 'Lock-on' feature means that the robot can make any movement in x and y axis while its yaw is fixed to certain target (usually the target enemy). As the robot's gimbal head angle is restricted to 180 degrees, this 'Lock-on' feature is essential for robot to shoot while performing any kind of movement. For example, the robot doesn't need to show its back (which is a dead-angle for its gun) or weak area to enemy to run away from that area.

Our Lock-on Algorithm uses Dynamic Window Approach to make trajectories but does not directly use the speed produced from it. It rather approximates the trajectory of DWA in quadratic scale with arbitrary angular velocity, thus enables our lock-on feature. DWA in three velocity space is only used to create and evaluate planar trajectory with certain curvature.

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \frac{1}{(dtw')^2 + 4} \begin{pmatrix} dt^2 w w' + 4 & 2 * dt(w' - w) \\ -2 * dt(w' - w) & dt^2 w w' + 4 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

Our quadratic approximation is superior to mere linear approximation as it takes account of the angular velocity which acts as an accelerative force to transversal speed thus curved trajectory can be made within each step. For example, when linearly approximated control loop of 10hz (0.1sec) with very fast angular velocity such as 5 rad/sec would have an error about

the scale of timestep*angular_velocity which is about 0.5 m/s which is 25% percent of robot's maximum speed. Thus, linear approximation for holonomic motion requires much faster control loop to safely follow curved trajectory which is infeasible. On the other hand, our algorithm can follow curved trajectory precisely and safely in high speed, slow control rate environments. Empirically, 10 hz control loop is just enough to drive safely while exploiting the robots maximum speed (2m/s). The approximated trajectory is undistinguishable from the original.

Our algorithm is superior to solely using DWA as ours can perfectly exploit the advantage of holonomic robots. Our algorithm is superior to TEB in that ours is much faster and more lightweight. TEB is an extremely heavy algorithm so it is not suitable for low-performance computing systems. TEB is especially bad when many obstacles exist. Time to time, the CPU and memory usage for calculation doubles, triples, and even quadruples, and eventually slows down the total robot system loop and enemy recognition or cause memory shortage. Although TEB yields very flexible and optimal trajectory, it seemed excessive for our system.

Open source ROS DWA planner package is used for trajectory generation.

v. Vision Recognition and Detection

Detection is the process of identifying an enemy and obtaining information from the identified data. The first thing to do in this process is to use deep-learning to identify the enemy in the image obtained from the camera. The algorithm used to identify the enemy is YOLO V3-TINY DNN. It used a combination layer of Darknet-53, which is the same structure as the Figure 9 below.

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figure 9 Convolution layer of Darknet-53

This was chosen to be fast enough to run real-time on embedded boards such as Jetson TX2, unlike other detection algorithms.

The following is how images are learned. No matter how good layer you use, you can never get a good image if the underlying image source is not good. Therefore, we learned from as many images as possible in various environments. A total of three environments (an outdoor environment with a white background, a dark background, and sunlight) were configured for optimum effect. We learned the number given to colors and armors. Figure 10 shows the learning process.

Figure 10 Detection learning process

A total of 14,515 images were labelled and configured for learning, and 64 images were studied 240,000 times per learning session. Average loss was advanced to 0.07.

The detection was detailed as above and was able to identify the enemy by 70% of the confidence up to 6 m. The identification of the number is correct up to 4 m, and then the two are confused. However, it is believed that learning in more images and in a more diverse environment will improve sufficiently. In fact, as the number of lessons increased to 90,000 > 170,000 > 240,000, it was possible to confirm that the number's identifiable distance increased noticeably to 1 m > 1.5 m > 4 m.

Detected robots are stored in memory as they are divided into allies robots, enemies 1 and 2. Of these, the armor is identified and stored in memory only for enemy robots. Through simple algorithms, the robot reduces the error of the detected robot and searches the armors inside the robot only for enemy robots. The robot is then selected to target the robot's most easily matched armor.

In addition, in the post process of additional data, the distance value is ousted using the Realsense Depth Camera D435. However, the images of the 640 * 480 resolution produced do not match each other because the specifications of the RGB camera and the Depth camera are different. Intel Corp.'s Librealsense SDK provides an align function to calibrate it, but when applied, it creates a serious problem that falls to 4~5 FPS. Therefore, we developed an algorithm to adjust the camera's picture angle through a direct experiment. To minimize computation, the linear approximation was used to match the depth pixel with the RGB pixel, and it was performed to find the true difference based on the time the distance was 3 m. This data is then sent to the blackboard and the gymbol controller.

All these systems worked as hard as possible to collect more than 10 FPS of detected data, taking into account that the ROS system clock was 10 Hz. Currently, 12 FPS can be obtained. As was prepared in Technical Proposal, tracking code using KCF was prepared in case the computation volume was increased in parallel with other processes. If used, the accuracy of the detector is slightly reduced, but the speed is fast, so it can be used to generate enough real-time if it falls below 10 FPS.

vi. Gymbol Control

Targeting and firing of moving targets were implemented using target armor data from the Detection node. The camera communicates commands for yaw, pitch angle to the gimbal via PID control based on the detected armor center pixel coordinates. Many experiments have determined p, i, d gain values for

the minimization of overshoots and quick rise time, and the offset values have been adjusted to enable accurate aiming through zero firing on the actual target. If the enemy is not detected in the camera's field of vision, it moves the gimbal left and right and looks around (sentry mode), aiming after detection, and firing when the target is within the error range of +/- 10 pixels. Figure 11 shows firing enemy's armor.

Figure 11 Firing projectile

Activate the sentry mode if the detection fails 10 frames in a row to prevent malfunction by entering the sentry mode when the detection is missed momentarily.

vii. Automatic Supply

Our automatic supply behavior starts by navigating to the reload zone and pre-align while moving. When it arrives to the zone, the robot fine-tunes its pose to align to the supplier. Currently, our navigation algorithm works just fine for alignment. However, PID control with LIDAR distance will be integrated to our system.

viii. Smart Decision

1. Using DQN(Deep-Q-Network)

The state of the agent was obtained using ROS and Python to learn reinforcement that can be applied to the actual robot. We have also created an environment in which action can take action. The ROS environment is configured in two locations, Stage and Gazebo, and similarly controlled by Python. Each robot agent receives its state from the blackboard, and immediately receives the previous before state and current state as input, producing the probability of 9 actions. The learning model is constructed based on the Deep-Q-Network (DQN), and the model is updated at the end of each episode. Figure 12 shows how it works.

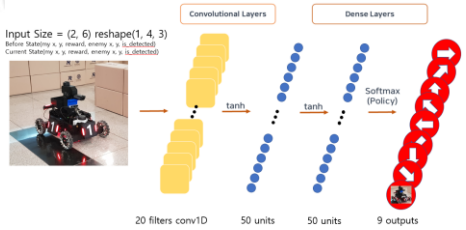

Figure 12 Deep-Q-Network model

The benefits of being able to apply directly to an environment that is directly linked to an ROS are very similar to the actual environment and has almost the same control methods. However, due to the difficulty of time acceleration and environmental cloning, they are not suitable for large amounts of learning. Therefore, we are also working on learning using the Unity Machine Learning Agent at the same time. Also, we

implement object detection on the vision of simulation robot model to enhance learning effectiveness by configuring simulation that is closer to the actual game environment during decision making through reinforcement learning.

2. Using Behavior Tree

Decision Tree strategically chooses what action to take for a given state. Using the state information, the robot determines the action and cooperates with ally robot.

A. Behavior

There are six types of actions that can be performed: fight, occupy defense zone, go to reload zone, escape, patrol, and sentry. Basically, for the cooperative movements, the behaviors cause the two robots to have the same motion. And Figure 13~15 represents each action.

i. Fight: Robots attack enemy by detecting their armor and shoot using projectile.

ii. Occupy defense zone: There are two robots in defense zone, one for occupying the defense zone and the other for covering the attack.

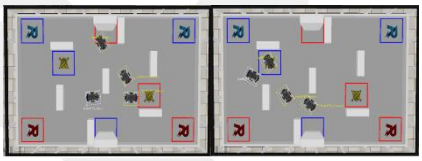

Figure 13 Fight and Occupy defense zone action

iii. Go to reload zone: Similar to the defense zone, it is divided into a robot that is being reloaded in the reload zone and a cover robot that covers the robot.

iv. Escape: A move that escapes a dangerous situation or evades elsewhere to do something else. This is divided into fast and slow escape depending on the state of the robot.

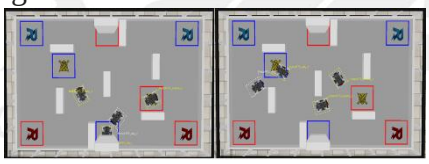

Figure 14 Reload zone and Escape action

v. Patrol: A movement that patrols to discover enemies or to create new situations. This is divided into active and passive patrol depending on the state of the robot.

vi. Sentry: It is a movement that moves the gimbal to the left and right (maximum +/- 90 degrees) to find the enemy in the present place.

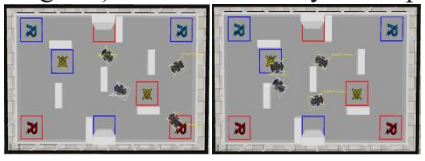

Figure 15 Patrol and Sentry action

B. Decision Tree

The robot uses a given state to determine what action to take through the decision tree. The Decision Tree determines the current situation according to four factors. The four factors are robot health, number of bullets, activation of defense zone, and detection of enemy robots.

The first node of the decision tree is the robot's Health power (HP). The reason is that the physical strength is the biggest factor determining the win / loss of the game, and the behavior style must be different according to the amount of HP. Therefore, the other nodes constituting the decision tree have different orders according to the amount of HP. The following Figure 16 is a schematic diagram showing decision tree.

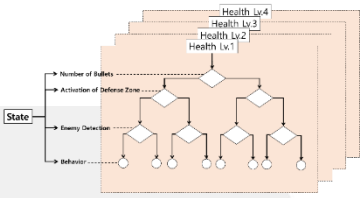

Figure 16 Schematic diagram of decision tree

C. Scheduling

During the game, the robot constantly determines what action to take for a given state. However, if time is set as in the case of activation of a defense zone, it is logical to move to the defense zone first. That is, a schedule is specified to perform a specific action for a predetermined time. You should make a behavior if predetermined behavior is not provided.

In case the robot cannot perform a desired decision unexpectedly, we put priorities in order to prepare various alternatives. The following Figure 17 is a schematic diagram of scheduling.

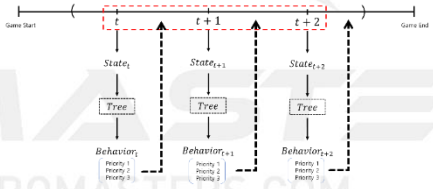

Figure 17 Schematic diagram of scheduling

C. Video Link

D. Reference

https://www.slamtec.com/en/Lidar/A2Spec
https://www.nvidia.com/ko-kr/autonomous-machines/embedded-systems-dev-kits-modules/
https://realsense.intel.com/depth-camera/#D415_D435
https://github.com/IntelRealSense/librealsense
https://darkpgmr.tistory.com/16
https://pjreddie.com/media/files/papers/YOLOv3.pdf
https://www.cs.toronto.edu/~vmnih/docs/dqn.pdf

[1]Unhelkar, Vaibhav V., et al. "Human-robot co-navigation using anticipatory indicators of human walking motion." *2015 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2015.

[2]Jin Yongzhu, “A study on human path prediction in human-robot collaboration with contactless vision sensor”. 2017