1. Task preview
2. Hardware
   1. Snapshort (整体方案)
   2. Robot Base Control
   3. Robot Gimbal Control
3. Software
   1. Environment
   2. Snapshort (整体方案)
   3. Framework
   4. Function Module
      1. Detection Module
      2. Localization Module
      3. Navigation Module
   5. Logic tree
4. References
5. **Task preview**

ss

**2. Hardware**

**2.1 Snapshort**

**2.2 Robot Base control**

**2.3 Robot Gimbal control**

**3. Software**

**3.1 Software Environment**

Consider the development time we have and the comprehensive requires to the robots of the competition, an excisting/open-sourse toolkits would be use during the development to avoid a bunch of repetitive works. So far we have abandoned the choice of building a framework oursleves, instead, choosing the Kinect version of ROS as the development environment for the host computer to build an algorithm framework.

Toolkits that require on the host development environment:

- OpenCV library (For visual processing, including armors detection, robot orientation recognition)

- CUDA toolkit (For speed up the GPU processing speed, that include visual and neural network processing)

- Localization and the Navigation modules (Other corresponding toolkits)

**3.2 Snapshort**

**3.3 Framework**

**3.4 Function Modules**

Function modules are the modules that processing the sensory input and transform the lower-level information to higher-level information for the host to take further actions.

**3.4.1 Detection module**

Visual detection is one of the most important method for autonomous robots in perceiving the environment. In this project, visual detection will be using on enemy identification, armors detection and obstacles highlight and so on.

We would use different algorithms on different objects detection.

The visual model of the armor is composed of two LED strips, each of them has the same length, parallel to each other and vertical to the ground.

The armor has two different colors, both of the brightness of blue and red are extremely high. Therefore, we need to extract the corresponding color blocks inside the image first.

Then, we are going to filter out the non-conforming color blocks according to the shape of the light bar, parallel features, length features, and direction features. We are going to match each other bar at the same time.

Lastly, filtered out all the remaining noise, leaving many pairs of color blocks that conform to the features

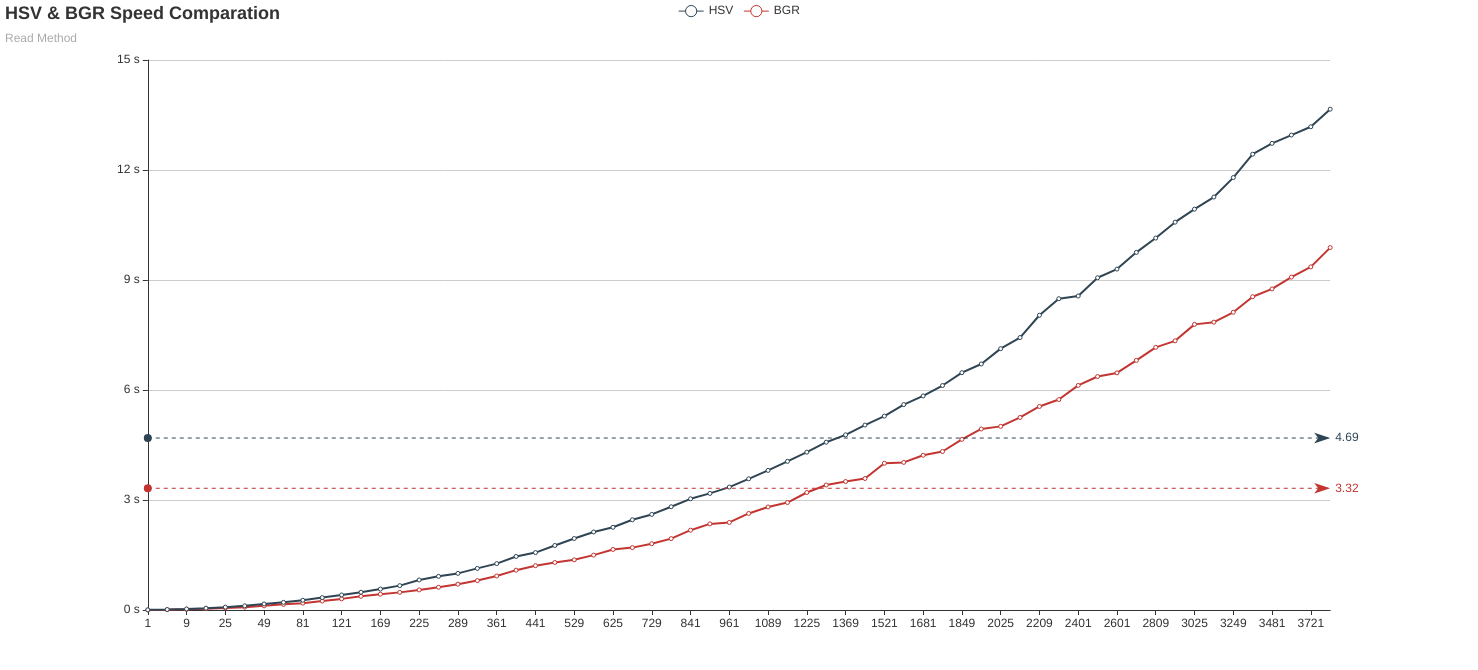


Armor(left)，Identifying armor(right)

We had already tested both algorithms based on RGB and HSV gamut extraction. Generally, the image storage in OpenCV defaults to BGR three-channel storage, the R and B for different channels.

The highest value for pure color and the remaining values are below 50 on average, which also proves the effectiveness of the RGB separation method.

By using well-design pixel pointer, we could achieve the fastest processor speed on OpenCV, so the RGB separation method should be considered more effective than HSV separation method.



In actual test, the color brightness and the purity are extremely high, which would cause the normal camera to be overexposed so that the imaging result of the center color of the light bar get white. Therefore, we would use an industrial camera with adjustable parameters. By lowering the brightness and exposure, increasing the contrast and slightly adjusting the saturation to optimize the algorithm on hardware level.

After the adjusting, the red color block recognition rate would greatly increase as well as the identification rate of armor.

**3.4.2 Localization module**

Localization is the process of determining where a robot is located with respect to its environment. It is one of the fundamental competencies required by an autonomous robot.

We are going to use AMCL algorithm, which known as Adaptive Monte Carlo Localization, this algorithm needs high precision odometer data for localization. Factors that leads to robot wheels slipping or idling could cause increase of odometer and by the time the cumulative error reaching a certain level, serious deviation of the positioning might occur, which leads to the robot’s inability to estimate its own position over time.

So, we are considering by adding a global localization system to help with the data correction, for that, we would like to use the Robomasters UWB positioning module. The previous test results from the DJI summer camp of the program of using this UWB positioning module as global localization system is not very satisfactory; the robot frequent collisions with obstacles during the dynamic navigation. It might because the module needs precisely calibration, further tests will be needed.

**3.4.3 Navigation module**

Path planning is one of the essential functions of autonomous mobile robots, which given the ability of a robot to find the shortest path between two points. As for the ICRA, the priority is to find out the path between robot and destination then get there as soon as possible, so we decided to use some of the classical path-planning algorithms this time

For this project, we are going to use A\* algorithm; A\* is an informed search algorithm that widely used in pathfinding and graph traversal, it finding the path by maintaining a tree of paths originating at the start node and extending those paths one edge at the time until its termination criterion is satisfied. The algorithm determines which of its paths to extend based on the cost of the path and an estimate of the cost required to extend the path all the way to the goal; then it selects the path that minimises.

Equation:

**f(n)=g(n)+h(n)**

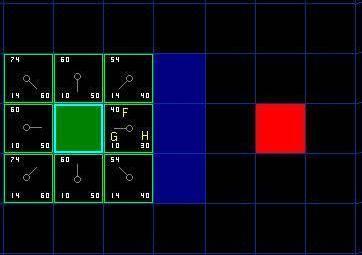
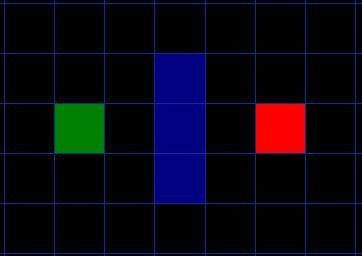
where n is the next node on the path, g(n) is the cost of the path from the start

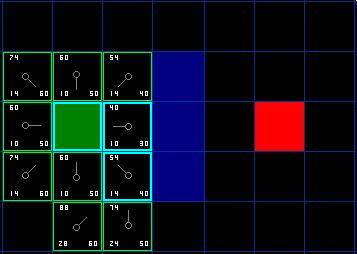
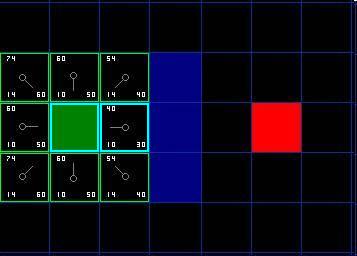
node to n and h(n) is a heuristic function that estimates the cost of the cheapest path from n to the goal.

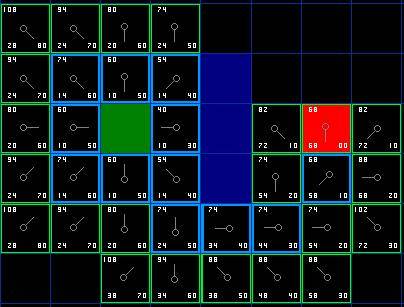
Pros of the algorithm: The obstacles and illegal terrains are highlights to reduce the unnecessary exploration, hence, increase the processing speed.

We are going to use the TEB local planner for local planning, TEB local planner able to optimize the robot trajectory online and produce alternative trajectories in distinctive topologies, moreover, it supports forward and backwards driving of robot, consider the mercurial environment of the battlefield(stage) the TEB would be the best choice for local planning.

**Planning processes:**







Assume that point A(green) wants to get to point B(red) and there is a wall(blue) separates the two points.

Firstly, algorithm add point A into an ‘open list'.

Secondly, it starts to look at all the reachable squares adjacent to the point inside the open list, ignoring squares with illegal terrain like walls. Add them to the open list too. For each of these squares, save point A as ‘parent square' which be used to trace our path later.

Lastly, algorithm move the starting square (point A) from the open list to ‘close list' after reachable squares were added into open list and start to determine which square to use when figuring out the path using the equation: f(n)=g(n)+h(n)

Where G is the movement cost to move from the given location to the goal square on the grid, following the path generated to get there. H is the estimated movement cost to move from the given square on the gird to the final destination.

Choice the square with the smallest f index and repeat step 1,2,3 until the final goal is inside an open list.

**3.5 Logic tree**

**Type of nodes in the logic tree:**

Selector node: Select the child-node from highest-priority to lowest-priority, once there are more than one child-node satisfied the condition, the child-node with the highest-priority will be chosen.

Condition node: Judging the condition, once it is not satisfied, return None, Otherwise points to the actions below the condition.

Action node: Specific function that controls robot action.

Abbreviation words in the logic tree:

FA\* Action means Frontal attack, both Master and Slave robots will start to frontal attack with the target.

SA\* Action means Sneak attack, robot will attack the target from behind.

FF\* Action means Focus fire, both Master and Slave robots will focus fire on one target.

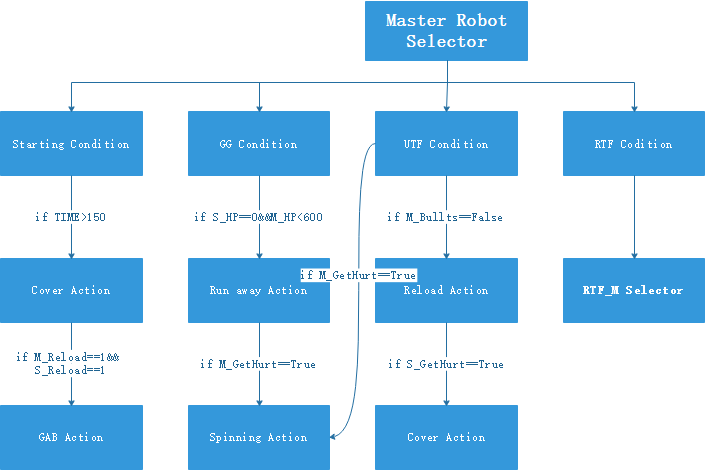
GAB\* Action means Go activate buff, robot will go and activate the buff.

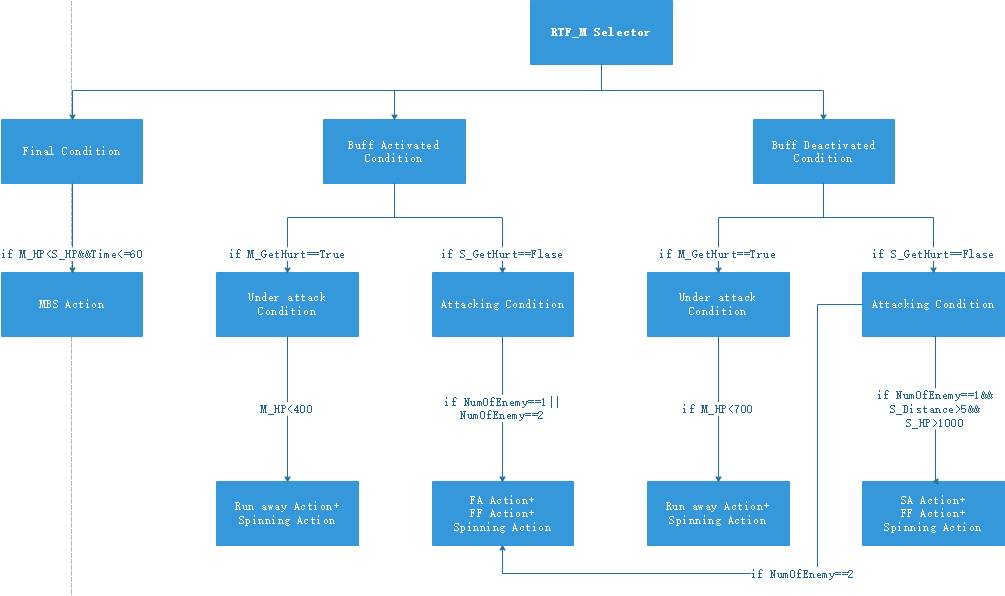
MBS\* Action means Master robot fellow behind the Slave robot.

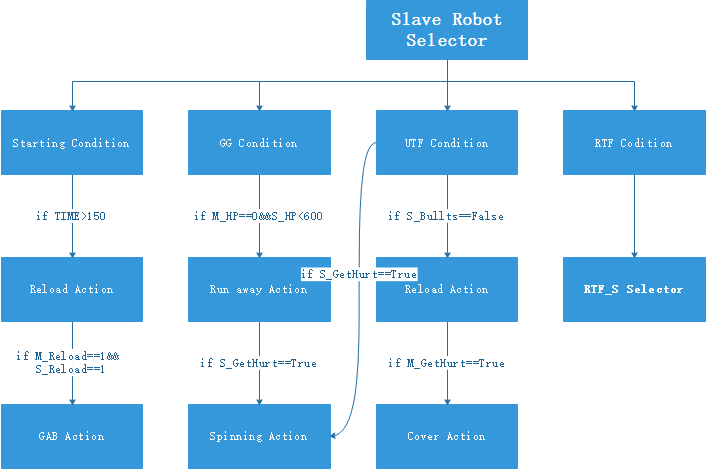
SBM\* Action means Slave robot fellow behind the Master robot.

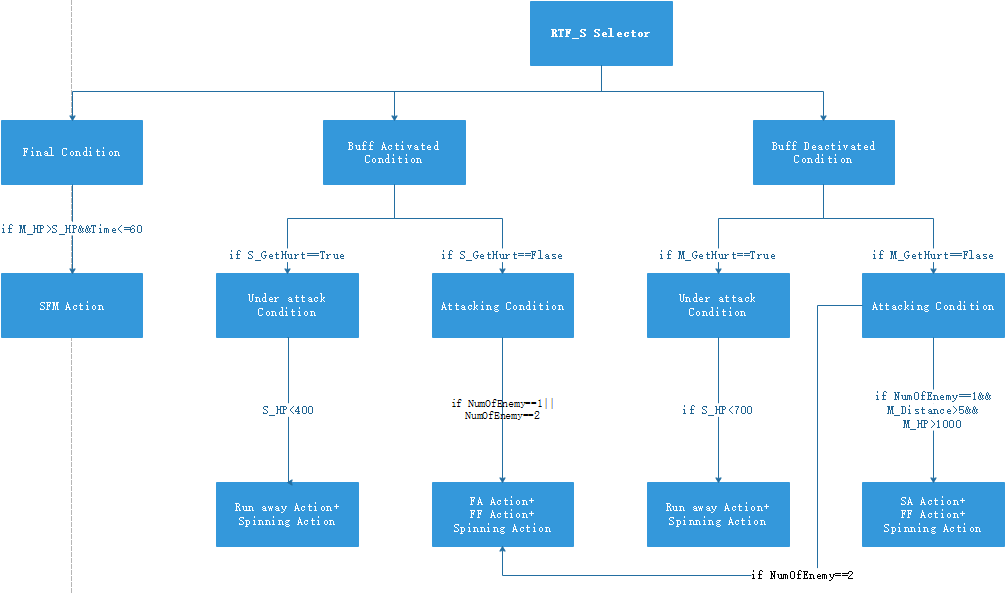
UTF\* Condition means unable to fire, robot is not loaded.

RTF\* Condition means ready to fire, robot is loaded.









1. Starting Condition

Judgment：time left greater than 150 second.

Behavior：Slave robot start reload and Master robot will cover it.

Effect：Robot will rush to reload after game starts.

1. GG Condition

Judgment：Only one robot surviving and the HP should below 600.

Behavior：Avoid contact, try to reserve.

Effect：Once there seems on chance for us to winning the game, robot will try to avoid contact as much as possible.

1. UTF Condition

Judgment：There is no bullet left inside the bullet tank.

Behavior：Reload

Effect：Robots will reload the bullet when the bullet tank is empty.

1. RTF Condition

Buff Activated Condition

Judgment：either one robot stayed in buff zone for more than 5 seconds.

Behavior：All robot will switch buff activated condition.

Effect：Robots will take more aggressive actions.

Buff Deactivated Condition

Judgment：Neither robot stayed in buff zone for more than 5 seconds.

Behavior：All robot will remain buff deactivated condition.

Effect：Robots will take normal actions.

Under attack Condition

Judgment：Any damage detected by referee system.

Behavior：Under attack Condition will last in the next eight second.

Effect：Robot would take action to avoid damage in this condition.

Attacking Condition

Judgment：Robot is fire on the target

Behavior：Attacking condition will last in the next three seconds.

Effect：Robot would choose different kind of way to attack the enemy.

**3. References**