Team Kinetic Robotics

Technical Proposal

Contents

1. Hardware 2

1.1 Framework 2

1.2 Mechanical part 2

1.2.1 Chassis module 3

1.2.2 Gimbal module 3

1.3 Device part 4

1.3.1 Slave computer 4

1.3.2 Host computer 5

1.3.3 Camera 6

1.3.4 LIDAR 6

1.3.5 Ultrasonic sensor 7

2. Embedded System 8

2.1 Framework 8

2.2 Chassis Control 9

3. Software 11

3.1 Software Environment 11

3.2 Framework 11

3.3 Function Modules 12

3.3.1 Detection & Recognition module 12

3.3.2 Localization module 13

3.3.3 Navigation module 14

3.4 Behavior tree 15

1. Hardware

1.1 Framework

We are going to use the standard robot as the RoboMaster AI Robot platform(hereinafter referred to as "Infantry") which developed by DJI TECHNOLOGY CO. Benefits of using Infantry includes high stability, low development costs. Cons like low personality, the bad mechanical design that would probably cut off the LIDAR scanning range also exist.

The hardware parts mainly include mechanical part and device part

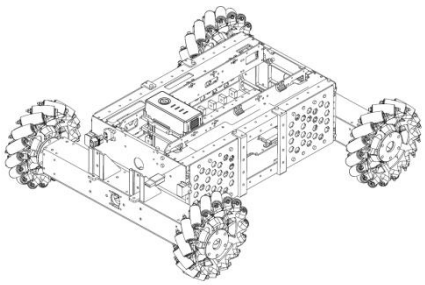
1.2 Mechanical part

The mechanical part mainly consists of two modules; Chassis module and gimbal module.

Module	Name	Quantity
Chassis module	M3508 P19 Brushless DC gear motor	4
	C620 Brushless Gear Motor Governor	4
	RM Mecanum wheel right	2
	RM Mecanum wheel left	2
Gimbal module	M2006 P36 Brushless DC Gear Motor	1
	GM6020 Brushless DC Motor	2
	C620 Brushless Gear Motor Governor	1
	Snail 2305 Racing Motor	2
	Snail 430-R Racing ESC	2

1.2.1 Chassis module

By using Mecanum wheels, Infantry is enabling active omnidirectional movement. Four 3508 Brushless DC gear motor with governors provide power to push the robot and payload. We will be using RoboMaster Development Board Type A(STM32F427IIH6) as the MCU(Slave computer).



Chassis Module

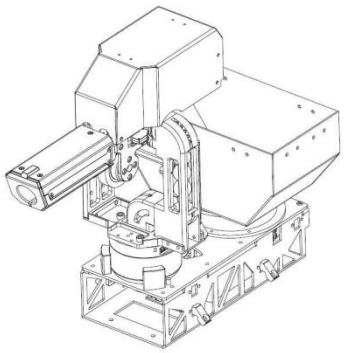
1.2.2 Gimbal module

Two dimensions gimbal drive by GM6020 DC brushless motor.

Loading system drive by M2006 P36 Brushless DC Gear Motor.

Firing system drive by DJI Snail 2305 Racing motor.

We will be using RoboMaster Development Board Type A(STM32F427IIH6) as the main control unit.



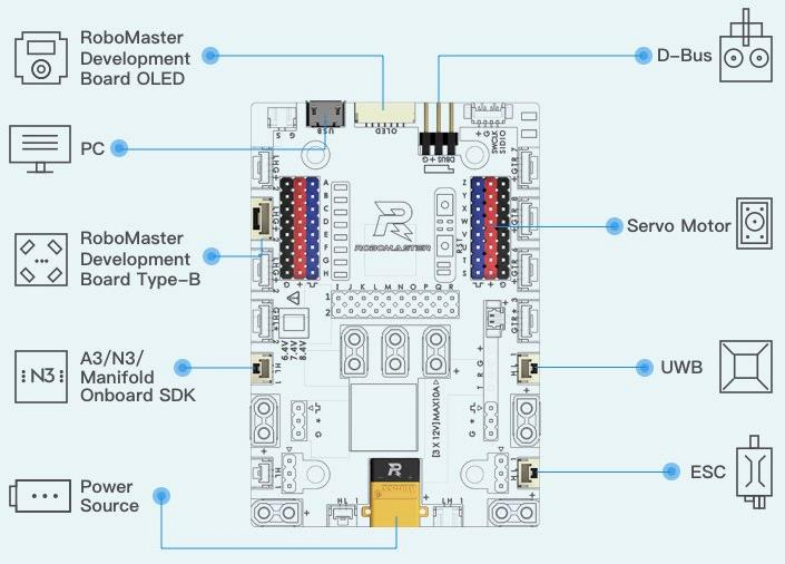
Gimbal Module

1.3 Device part

1.3.1 Slave computer

Device	Name	Quantity
Slave computer	RM Development Board Type A	2
Host computer	Intel 7th, i5 IPC	1
	Nvidia Jetson TX2	1
Camera	High speed 500-resolution IC	1
LIDAR	YDLIDAR G4	1
Ultrasonic sensor	HY-SRF05	4
Battery	TB47D Battery	1

Two RM development board type A will be used as slave-computer on chassis module and gimbal module separately. It features various ports for further development such as UART port and CAN port that could allow it communicate with the master computer, PWM port that could drive servo and brushless motors, and it also supports UWB localization and SDK development. Multiple protection mechanisms including a built-in ESD on PWM interfaces prevent the reverse connection, over-voltage, and over-current.



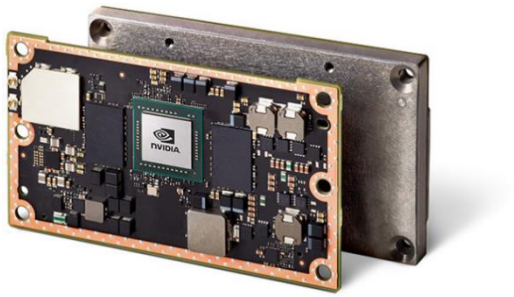
RM Development Board Type A

1.3.2 Host computer

So far we have two choices for master computer, we could either directly buy the Nvidia TX2, it is expensive but will save much time or just DIY an IPC, which will be cheaper than the previous choice but time-consuming.

For the first choice, Jetson TX2 is a tiny little board built around Nvidia Pascal-family GPU with 256 CUDA cores which means faster speed on matrix multiplication. The CPU complex consists of a Quad-core A57 ARM processor connected to a dual core Denver processor, way slower than the I7, but it is ok to use.

	Nvidia Jetson TX2
GPU	Nvidia Pascal, 256 CUDA cores
CPU	HMP Dual Denver 2/2 MB L2
	Quad ARM A57.2 MB L2
Memory	8 GB 128bit LPDDR4 59.7GB/s
Data Storage	32GB eMMC



Jetson TX2

For the DIY IPC choice, we would use the chassis with the dimension of 18.5X4.5X19.75CM(W*H*D) to meet the Infantry mechanical space where chassis are going to be located.

IPC modules	Name	Quantity
CPU	Intel 7th i5	1
Mainboard	ASUS H110T	1
SDRAM	4G DDR4	2
Hard disc	128G SSD	1

IPC Chassis

We are going to remain both choices of the master computer, only because ROS requires a higher CPU performance, so that it would be better if we develop on IPC and then transfer it to the TX2 after the developed, it is going to save a lot of time.

1.3.3 Camera

Based on our previous experiences, it is highly imported to have a camera that can reduce motion blur. What's more, the parameter adjustment is also one of the necessary functions. And the convenience.

As mentioned above we finally chose a camera that supports USB3.0, parameters adjustment and high frame rate for motion blur reduction. The resolution of the camera is not that satisfactory, with only 640x480 resolution rate, but the competition does not require the resolution rate that much. It is quite enough to get image features such as amour light bar on this resolution rate. In short, the camera we have chosen is quite enough for the competition.

Camera

1.3.4 LIDAR

LIDAR is a critical device for localization and navigation

Since the LIDAR works like shine a small light at a surface and measure the time it takes to return to its source, Therefore the adaptive scanning frequency and range sample frequency would be the high priorities for choosing a LIDAR.

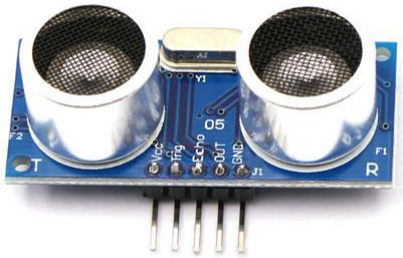
The LIDAR we are going to use is the G4 LIDAR which developed by YDLIDAR CO, with range sample frequency about 9000hz, scanning range around 16m and maximum 12Hz adaptive scanning frequency. The reason why we are choosing this LIDAR is that it has the highest adaptive scanning frequency and range sample frequency among the same type of product, and the price is also quite reasonable.

LiDAR

1.3.5 Ultrasonic sensor

Ultrasonic sensor will be used as an auxiliary positioning device of compensation and calibration for LiDAR positioning during the robot reloading process. HY-SRF05 ultrasonic sensor can work from 2cm to 3m with 15 degrees measuring the angle and ranging accuracy is reaching no more than 3mm.

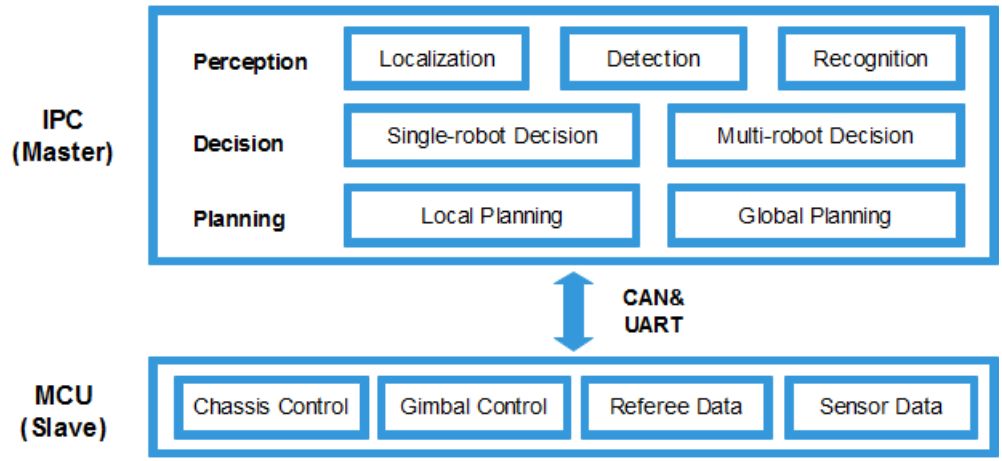
Since it is a relatively mature product on the market, therefore we chose it as an auxiliary positioning device.



Ultrasonic Sensor

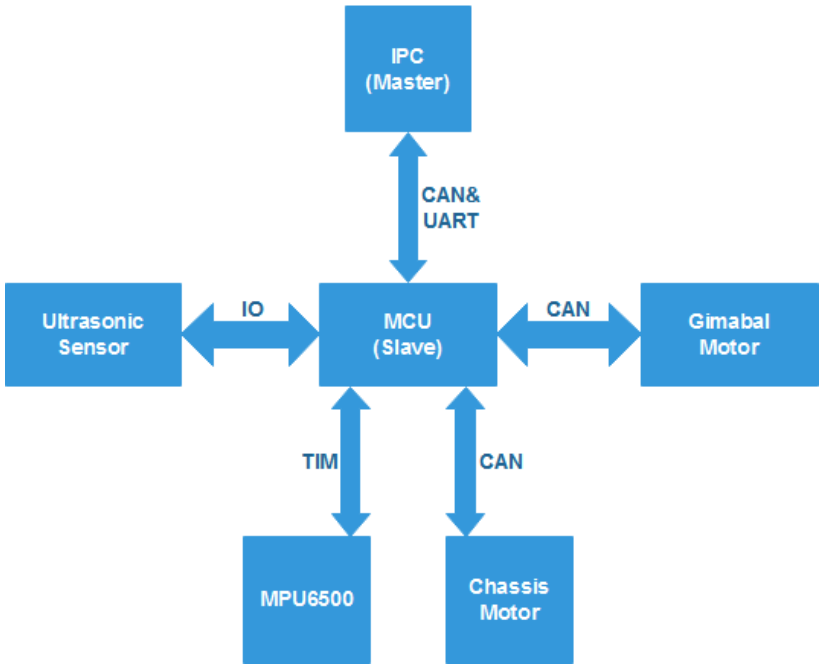
2. Embedded System

2.1 Framework



Embedded system framework on above.

MCU (etc. RM Development Board) stands as slave computer and responses for lower level data processing, that includes necessary motion control(etc. gimbal, chassis), transfer referee system data and sensors data to master robot.



MCU Communication Framework

We are going to use CAN1 to communicate with chassis module motors and gimbal module motors because CAN is faster than UART in this situation and besides that, CAN is more comfortable coding compare with UART.

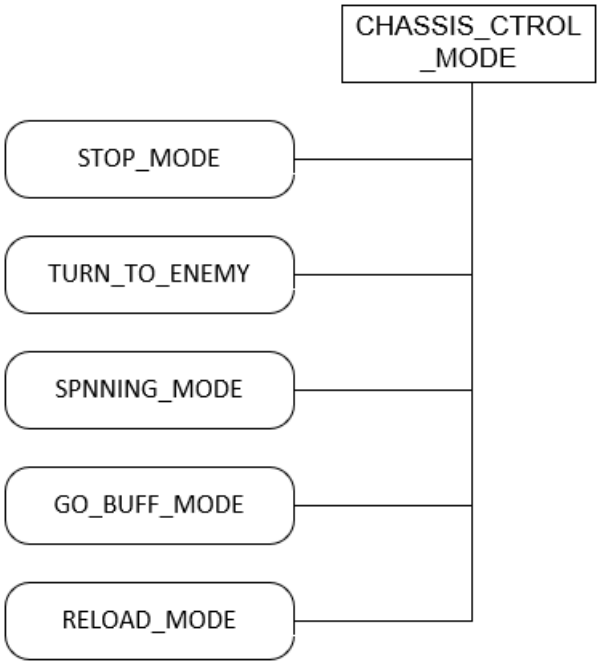
There are few basic functions we have to realize.

Reloading: The robot must be able to reload bullets.

Chasing: Chassis and gimbal must be able to fellow the target once the target has been detected.

Spinning: The robot will try as much as it can to avoid enemy attacks by taking spinning motion.

2.2 Chassis Control



Chassis control model

STOP_MODE:

Judgment：When received the order from master computer.

Behavior：Chassis module and gimbal module locked immediately.

Effect：Robot gets into sleep mode and wait for further order.

TURN_TO_ENEMY:

Judgment：If there is damage detected by the referee system.

Behavior：Chassis turns to the direction of armor attacked immediately.

Effect：Robot turns to the direction of armor attacked immediately.

SPNNING_MODE:

Judgment：If there is damage detected by the referee system

Behavior：Chassis starts spinning immediately to avoid damage.

Effect：Reducing the damage.

GO_BUFF_MODE:

Judgment：When received the order from the master computer.

Behavior：Robot rush to the buff zone.

Effect：Reducing the damage.

RELOAD_MODE:

Judgment：When bullet tank is empty.

Behavior：Robot rush to the reloading zone and starts to reloading.

Effect：Reloading bullet.

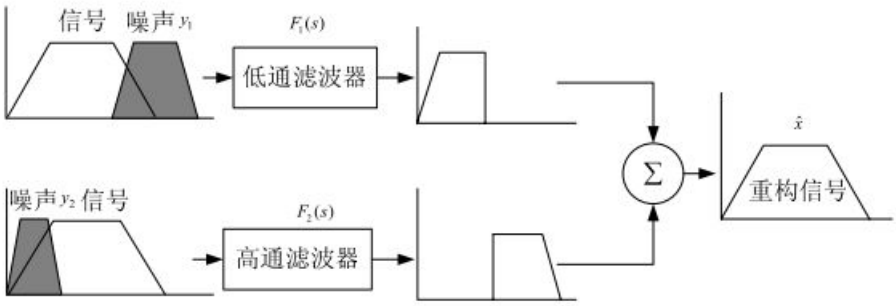
Closed-loop system will be using on both the DC motor's position and speed control; position loop will be the outer loop, the target input per cycle is the expectations of the encode step size per cycle. Integrate the expectation and the actual value every period on position loop so that the error could be compensated to the next cycle's expectation. A Kalman filter will be used on the speed loop sampling process to smooth the velocity value, makes the speed control smoother and more accurate.

2.3 Gimbal Control

Gimbal and chassis control will be using the same development board which locates on the gimbal.

Closed-loop system will be also using on DC motor's position and speed control.

An MPU6500 takes charge of pitch axis and yaw axis data sampling.



Firstly, using a low-pass filter on inertial navigation data sampling

Secondly, Using Mahony filter and quaternion to integrate and calculate the acceleration and angular velocity at 500Hz to get an accurate gimbal attitude.

3. Software

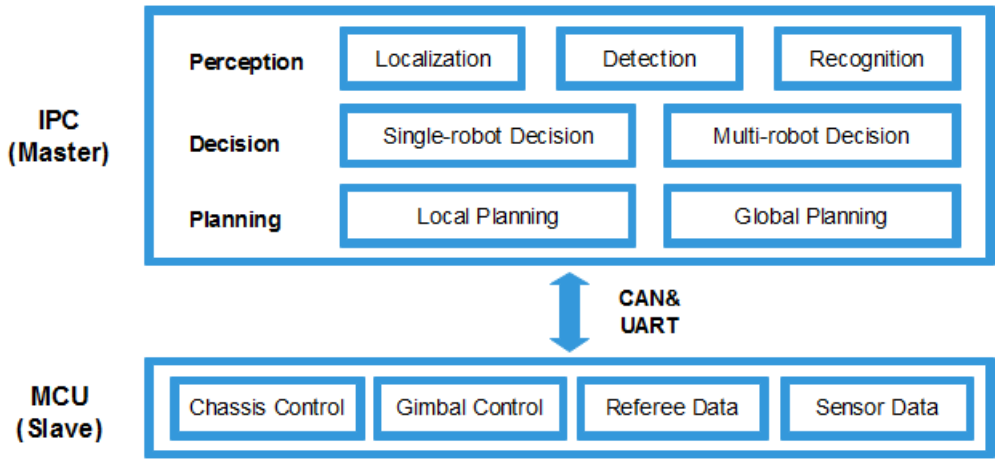
3.1 Software Environment

Consider the development time we have and the comprehensive requires to the robot of the competition, an exciting/open-source toolkits would be used during the development to avoid a bunch of repetitive works. So far, we have abandoned the choice of building a framework ourselves, instead, choosing the Kinect version of ROS as the development environment for the host computer to construct an algorithm framework.

TODO:

- Pygame environment (For game simulation)
- 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 Framework



Software framework (includes Embedded system) on above.

IPC (etc. Nvidia TX2) stands as a master computer and takes the change of higher level calculation such as object detection, object relative distance calculation

Perception including:

Localization: that contents robot localization, object detection, and object recognition.

Decision: base on the behavior tree or other machine learning framework.

Planning: global path planning and local path planning.

3.3 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.

3.3.1 Detection & Recognition module

Visual detection is one of the most important methods for autonomous robots in perceiving the environment. In this project, visual detection is 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.

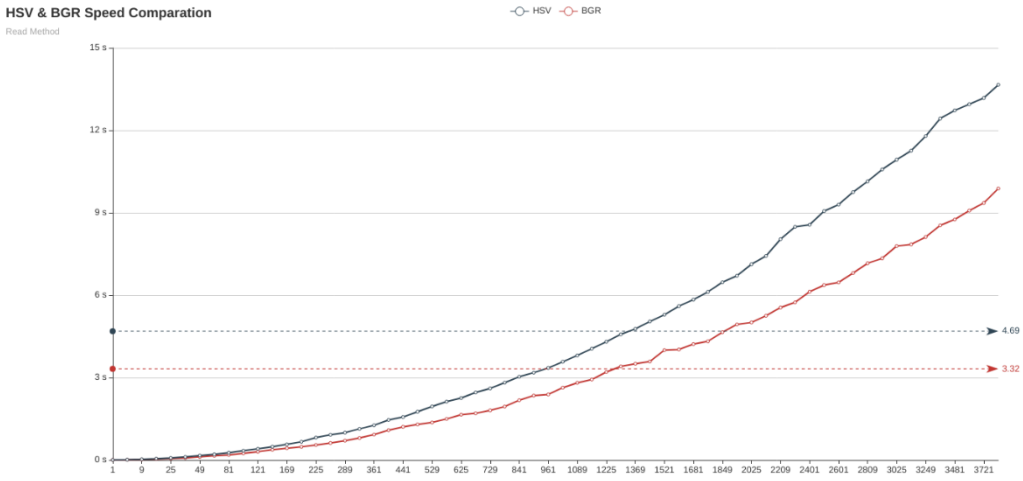
Lastly, filtered out all the remaining noise, leaving many pairs of color blocks that conform to the features.

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.



HSV vs BGR

Test results show that the RGB separation method has higher efficiency and higher success rate compare with the HIS separation method results.

In the actual test, the color brightness and the purity are incredibly high, which would cause the regular 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 the hardware level.

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

3.3.2 Localization module

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

We are going to use the AMCL algorithm, which known as Adaptive Monte Carlo Localization, this algorithm needs high precision odometer data for localization. Factors that lead to robot wheels slipping or idling could cause an 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 position over time.

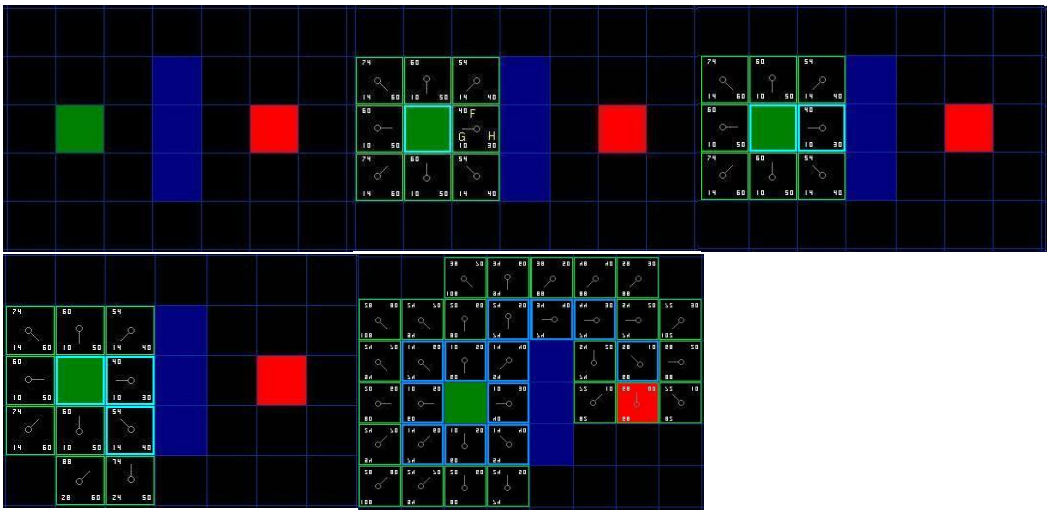
So, we are considering by adding a global localization system to help with the data correction, for that, we would like to use the RoboMaster UWB positioning module. The previous test results from the DJI summer camp of the program of using this UWB positioning module as global localization system is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

3.3.3 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.

This time, we are going to use A* algorithm; it is an informed search algorithm that widely used in pathfinding and graph traversal, it finds 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 minimizes.

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 backward 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, the algorithm adds the 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.

3.4 Behavior tree

Type of nodes in the behavior 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 dual-core 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 behavior tree:

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

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

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

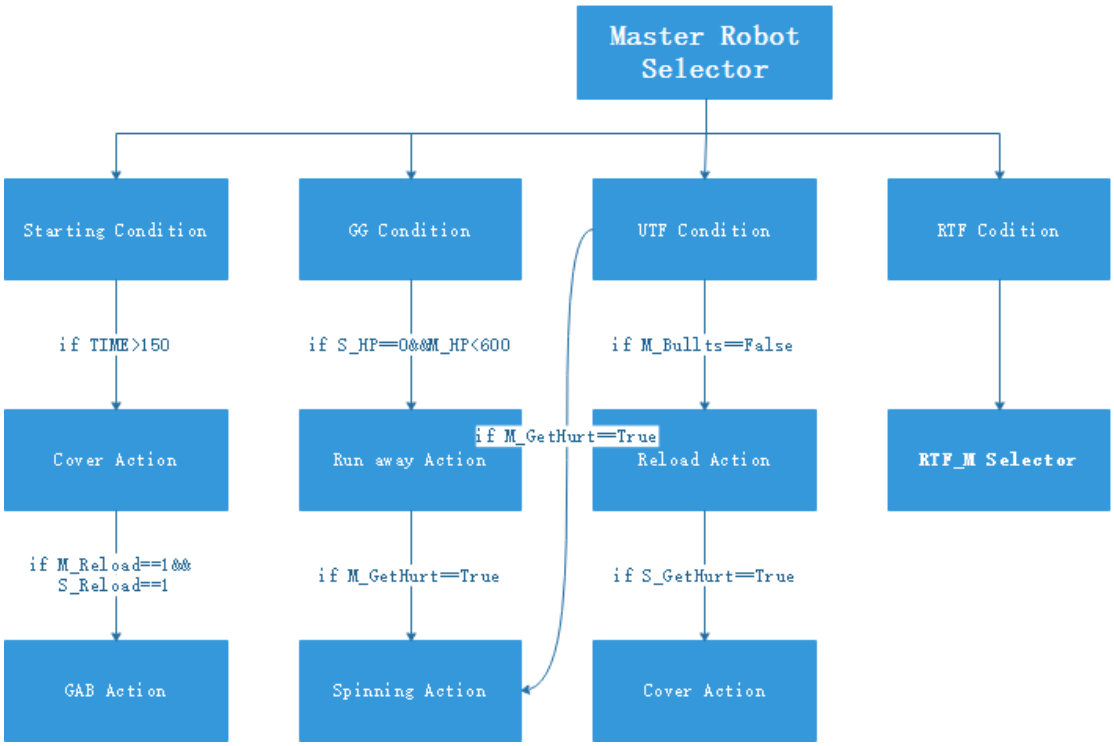
GAB* Action: Go and activate buff, the robot will go and activate the buff.

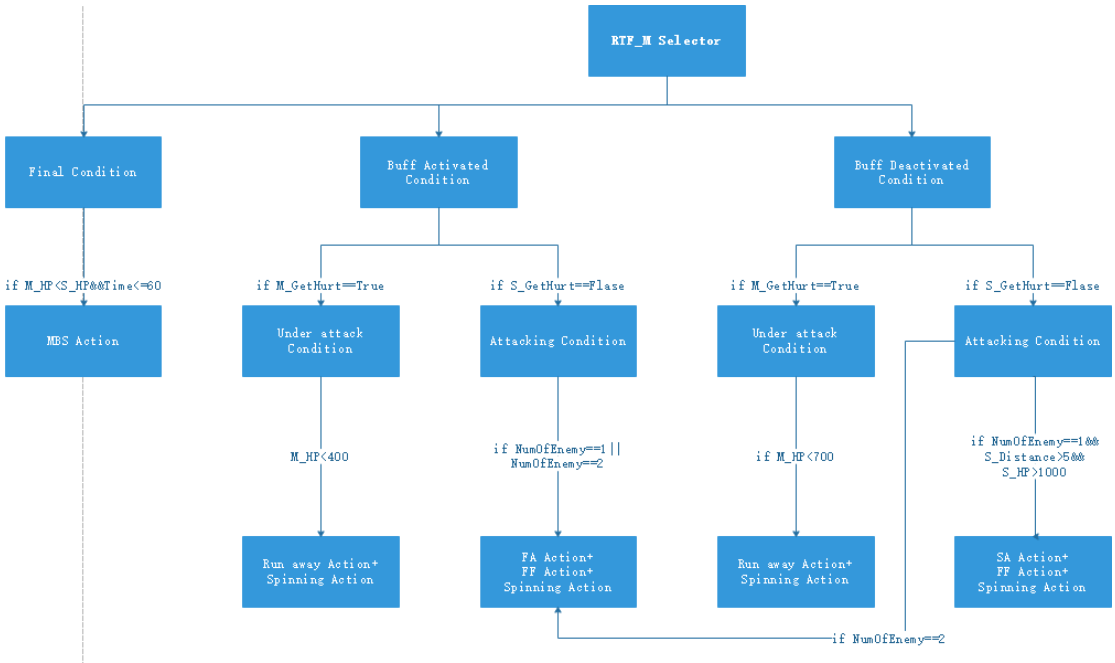
MBS* Action: Master robot fellow behind the Slave robot.

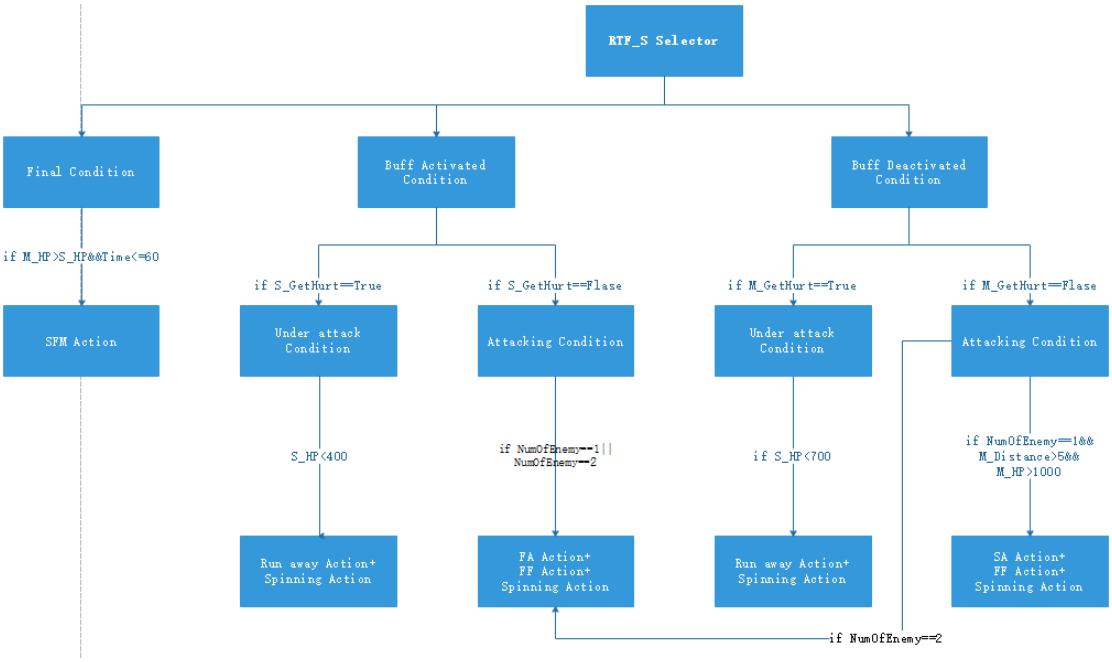
SBM* Action: Slave robot fellow behind the Master robot.

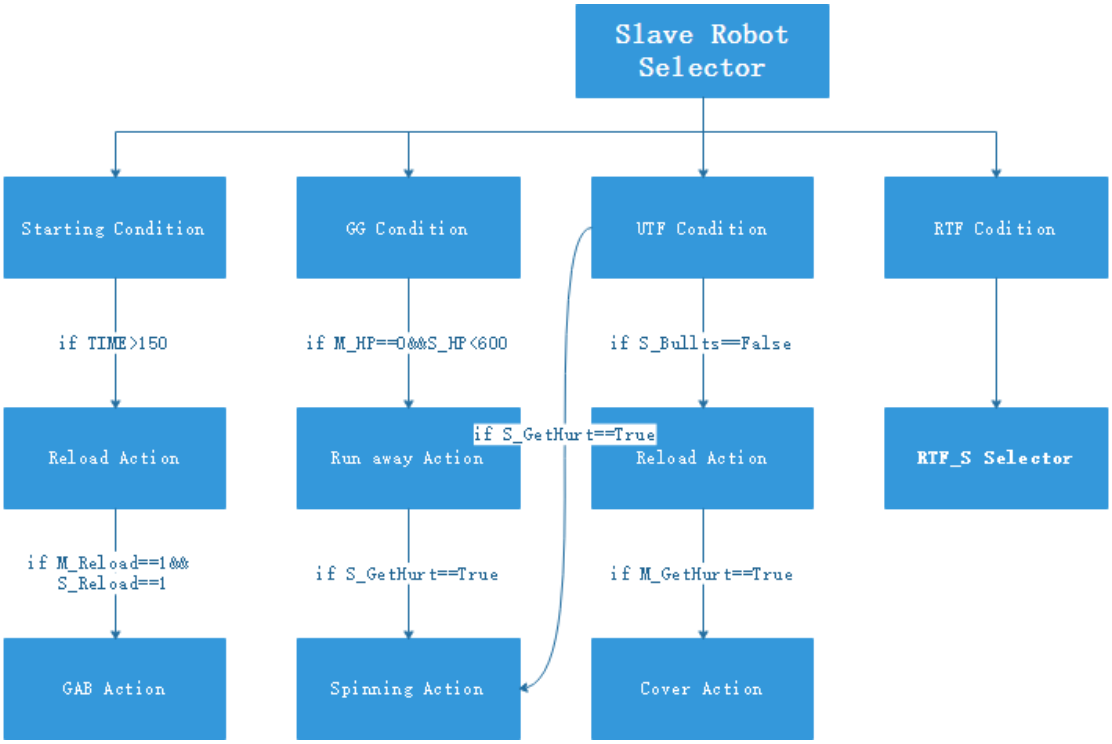
UTF* Condition: unable to fire, robot is not loaded.

RTF* Condition: ready to fire, the robot is loaded.


Master Robot Selector
Starting Condition
GG Condition
UTF Condition
RTF Codition
if TIME>150
if S_HP==0&&M_HP<600
if M_Bullts==False
Cover Action
Run away Action
Reload Action
RTF_M Selector
if M_GetHurt==True
if M_Reload==1&&
S_Reload==1
if M_GetHurt==True
if S_GetHurt==True
GAB Action
Spinning Action
Cover Action


RTF_M Selector
Final Condition
Buff Activated Condition
Buff Deactivated Condition
if M_HP<S_HP&&Time<=60
if M_GetHurt==True
if S_GetHurt==Flase
if M_GetHurt==True
if S_GetHurt==Flase
MBS Action
Under attack Condition
Attacking Condition
Under attack Condition
Attacking Condition
M_HP<400
if NumOfEnemy==1 ||
NumOfEnemy==2
if M_HP<700
if NumOfEnemy==1&&
S_Distance>50&&
S_HP>1000
Run away Action+
Spinning Action
FA Action+
FF Action+
Spinning Action
Run away Action+
Spinning Action
SA Action+
FF Action+
Spinning Action
if NumOfEnemy==2


RTF_S Selector
Final Condition
Buff Activated Condition
Buff Deactivated Condition
if M_HP>S_HP&&Time<=60
if S_GetHurt==True
if S_GetHurt==Flase
if M_GetHurt==True
if M_GetHurt==Flase
SFM Action
Under attack Condition
Attacking Condition
Under attack Condition
Attacking Condition
S_HP<400
if NumOfEnemy==1||NumOfEnemy==2
if S_HP<700
if NumOfEnemy==1&&M_Distance>5&&M_HP>1000
Run away Action+Spinning Action
FA Action+FF Action+Spinning Action
Run away Action+Spinning Action
SA Action+FF Action+Spinning Action
if NumOfEnemy==2


Slave Robot Selector
Starting Condition
GG Condition
UTF Condition
RTF Codition
if TIME>150
if M_HP==0&&S_HP<600
if S_Bullts==False
Reload Action
Run away Action
Reload Action
RTF_S Selector
if S_GetHurt==True
if M_Reload==1&&S_Reload==1
if S_GetHurt==True
if M_GetHurt==True
GAB Action
Spinning Action
Cover Action

1. Starting Condition

Judgment：time left greater than 150 second.
Behavior：Slave robot starts to reload, and Master robot will cover it.
Effect：Robot will rush to reload after game starts.

2. GG Condition

Judgment：Only one robot is surviving, and the HP should come below 600.
Behavior：Avoid contact, try to reserve.
Effect：Once there seems one chance for us to winning the game; robot will try to avoid contact as much as possible.

3. 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.

4. RTF Condition

Buff Activated Condition
Judgment：Either one robot stayed in the 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 the 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：If there is damage detected by the 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 a different kind of way to attack the enemy.