RM2016Camp 第五组嵌入式设计资料

空中机器人抓取机构控制

任务要求：

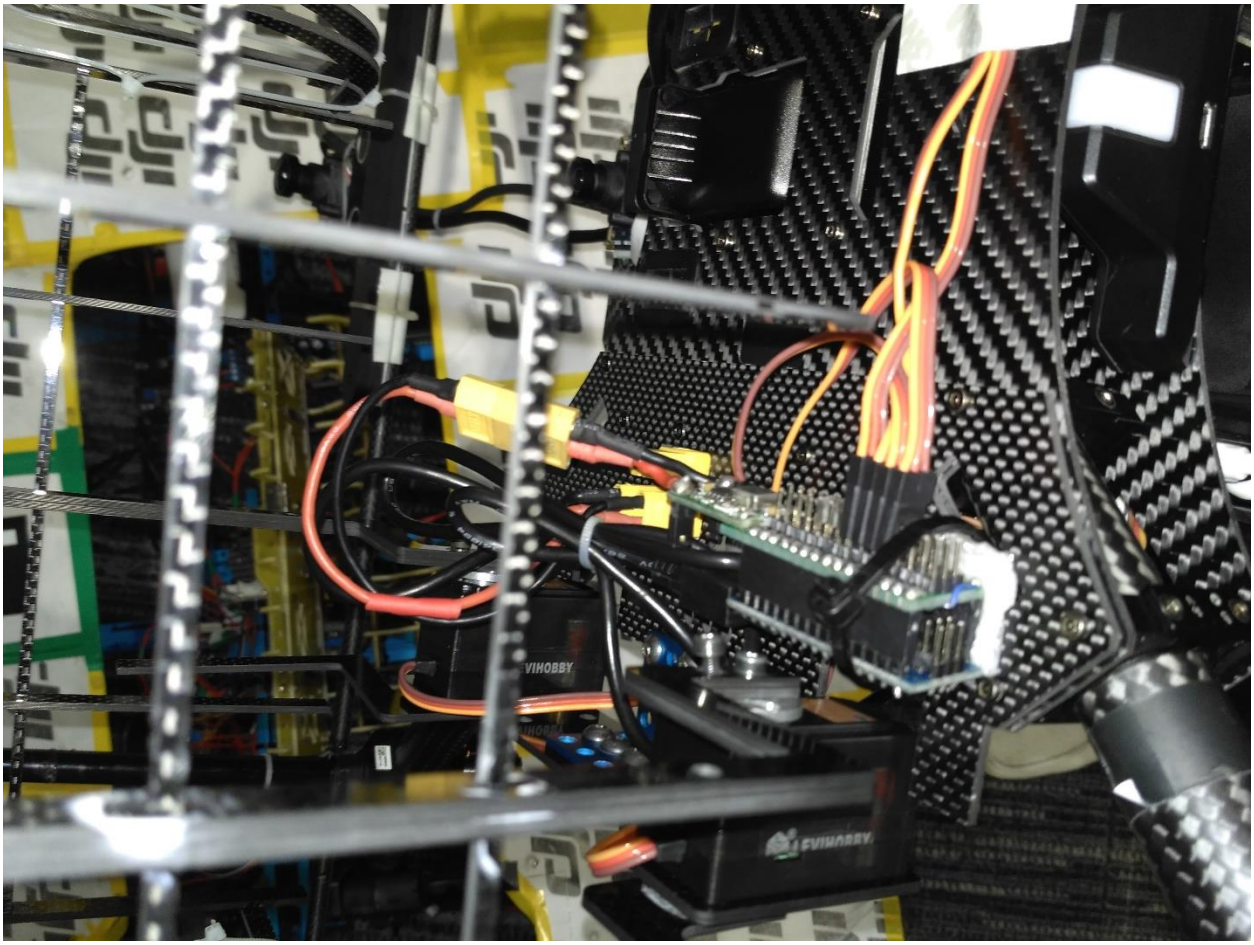
1、能通过串口和飞机通信，接收来自机载电脑的指令，分别控制两个机械爪的开合状态，每个机械爪两个标准舵机，共四个舵机。

2、控制器要尽量轻盈，体积尽量小。

方案概述：

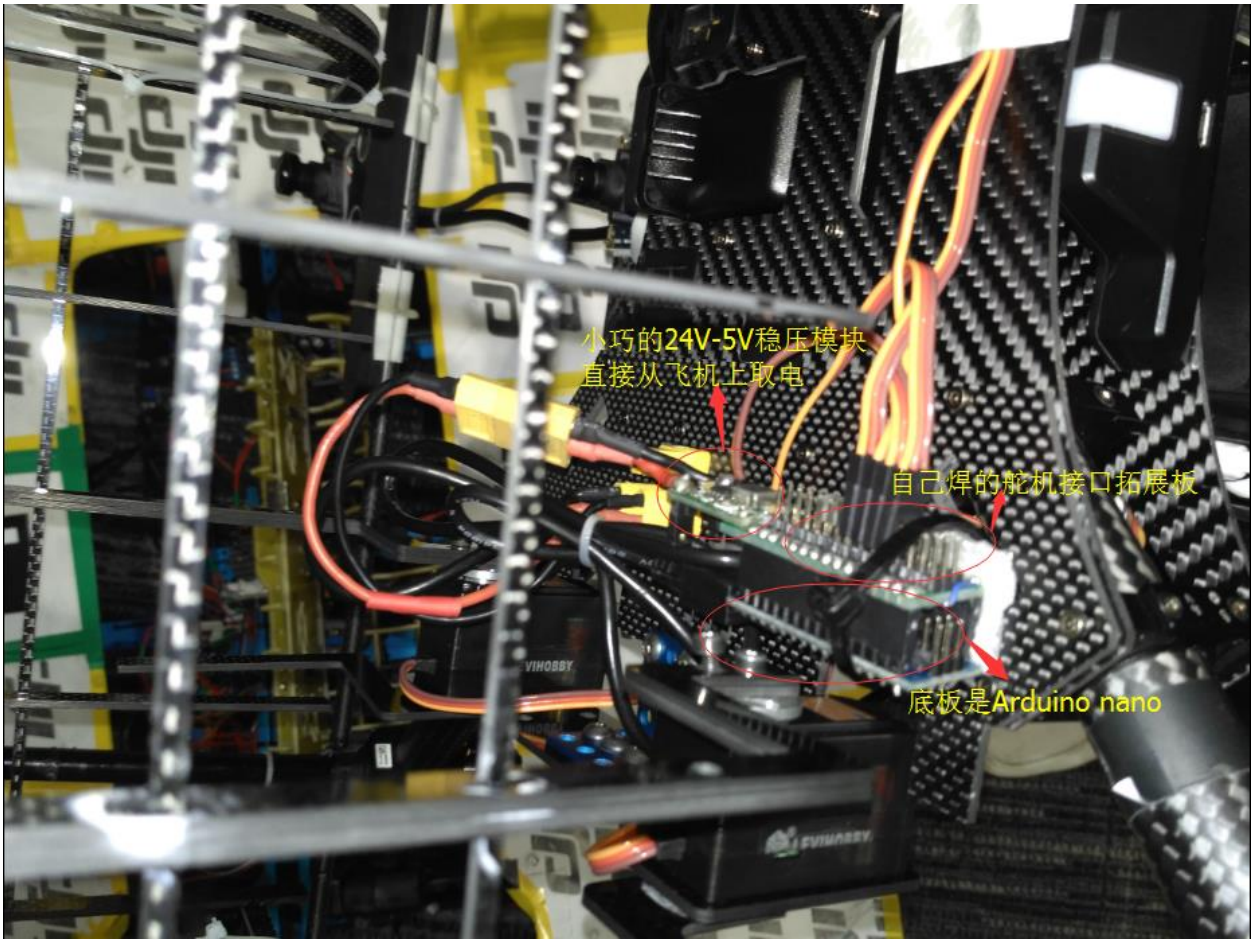
Arduino nano + 自制多路舵机拓展板 + 轻巧 5V 稳压（用于直接从飞机上取电）

方案图示：

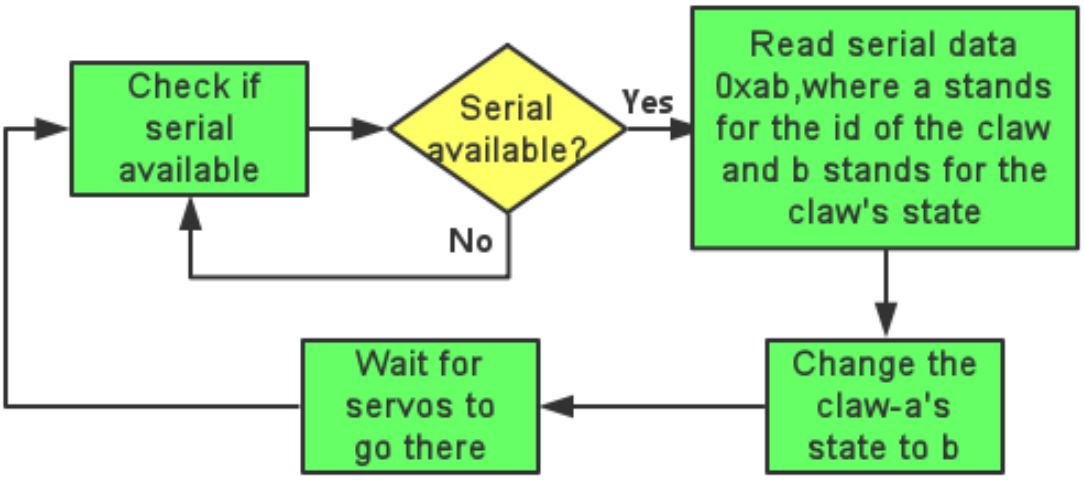


用 Arduino nano 做的机械爪控制器，重量轻盈，体积小巧，背面有多达 10 路的 PWM 信号输出接口，可同时控制 10 路舵机；自带 22.4V 转 5V 稳压模块，直接从飞机上取电，为飞行任务量身定制

硬件图解：



软件流程：



Claw for Aerial Robot Work Flow
(By Bangjie Mo)

Arduino nano 实现代码：

```
#include <Servo.h>

#define CLAW_NUM 5      //爪子数量
#define SERVO_NUM 2     //每个爪子所使用的舵机数量

#define CLAW_CLOSE_ANGLE_OFFSET 0    //爪子闭合时的角度偏移（为了补偿机械误差）
#define CLAW_OPEN_ANGLE_OFFSET 35    //爪子张开时的角度偏移（为了补偿机械误差）
#define CLAW_OFFSET -10              //爪子整体的角度偏移（为了不挡住摄像头）
#define CLAW_OPEN_ANGLE_DEFAULT {0+CLAW_OPEN_ANGLE_OFFSET, 180-
CLAW_OPEN_ANGLE_OFFSET-10}  //爪子张开时两个舵机的角度预设
#define CLAW_CLOSE_ANGLE_DEFAULT {90+CLAW_OFFSET, 90+CLAW_OFFSET} //爪子关闭时两个舵机
的角度预设

/*****************************************************************************/
/* 指令格式 0xab, 高四位 a 代表爪子 id(1~4),低四位表示该爪子的状态（0：关，1：开） */
/*****************************************************************************/
const uint8_t CLAW_CLOSE_CMD[CLAW_NUM] = {0x10, 0x20, 0x30, 0x40, 0x50}; //控制爪子关闭的指
令集合
```

```
const uint8_t CLAW_OPEN_CMD[CLAW_NUM]  = {0x11, 0x21, 0x31, 0x41, 0x51};/ /控制爪子张开的指令集合

/*****************************************/
/* 用查表的方式来确定舵机的角度 */
/*****************************************/
const uint8_t CLAW_OPEN_ANGLE[CLAW_NUM][SERVO_NUM]  = {CLAW_OPEN_ANGLE_DEFAULT, CLAW_OPEN_ANGLE_DEFAULT, CLAW_OPEN_ANGLE_DEFAULT, CLAW_OPEN_ANGLE_DEFAULT, CLAW_OPEN_ANGLE_DEFAULT};
const uint8_t CLAW_CLOSE_ANGLE[CLAW_NUM][SERVO_NUM] = {CLAW_CLOSE_ANGLE_DEFAULT, CLAW_CLOSE_ANGLE_DEFAULT, CLAW_CLOSE_ANGLE_DEFAULT, CLAW_CLOSE_ANGLE_DEFAULT, CLAW_CLOSE_ANGLE_DEFAULT};

Servo servo[CLAW_NUM][SERVO_NUM]; //控制对象：舵机声明，数量 CLAW_NUM*SERVO_NUM
uint8_t pin[CLAW_NUM][SERVO_NUM] = {{2, 3}, {4, 5}, {6, 7}, {8, 9}, {10, 11}}; //声明舵机连接的 IO 口
uint8_t led = 13; //LED 接口，用来指示爪子开关状态

/* 开爪函数 */
void OpenClaw(int i)
{
  for(int j = 0; j < SERVO_NUM; j++)
    servo[i][j].write(CLAW_OPEN_ANGLE[i][j]);
}

/* 闭爪函数 */
void CloseClaw(int i)
{
  for(int j = 0; j < SERVO_NUM; j++)
    servo[i][j].write(CLAW_CLOSE_ANGLE[i][j]);
}

/* 点亮 LED 灯 */
void LED_ON()
{
  digitalWrite(led, HIGH);
}

/* 熄灭 LED 灯 */
void LED_OFF()
{
  digitalWrite(led, LOW);
}

/* LED 灯状态翻转 */
void LED_TOGGLE()
{
  digitalWrite(led, !digitalRead(led));
}
```

```
/* 指令处理 */
void process(uint8_t cmd)
{
  for(int i = 0; i < CLAW_NUM; i++)
  {
    if(cmd == CLAW_OPEN_CMD[i])
    {
      OpenClaw(i);
      LED_ON();
      Serial.write(cmd);
      delay(10);
    }
    if(cmd == CLAW_CLOSE_CMD[i])
    {
      CloseClaw(i);
      LED_OFF();
      Serial.write(cmd);
       delay(10);
    }
  }
}

/* 初始化舵机 */
void SetupServo()
{
  for(int i = 0; i < CLAW_NUM; i++)
  {
    for(int j = 0; j < SERVO_NUM; j++)
    {
      servo[i][j].attach(pin[i][j]);
      OpenClaw(i);
    }
  }
}

/* 初始 LED */
void SetupLed()
{
  pinMode(led, OUTPUT);
}

/* 全局初始化 */
void setup()
{
  Serial.begin(115200);
  SetupServo();
  SetupLed();
  Serial.println("setup done!\n");
}

/* 任务循环 */
void loop()
{
```

```
  if(Serial.available()) //检查串口是否有数据到来
  {
    uint8_t cmd = Serial.read(); //读取指令
    process(cmd); //处理指令
  }
}
```

地面机器人控制

任务要求：

Mission 1（移动模式）：地面机器人能自动从 1 区走到 3 区，并且避开移动篮筐

Mission 2（停机坪模式）：地面机器人从 3 区抓取公仔，放入自身储物仓，等待飞机降落抓取

方案概述：

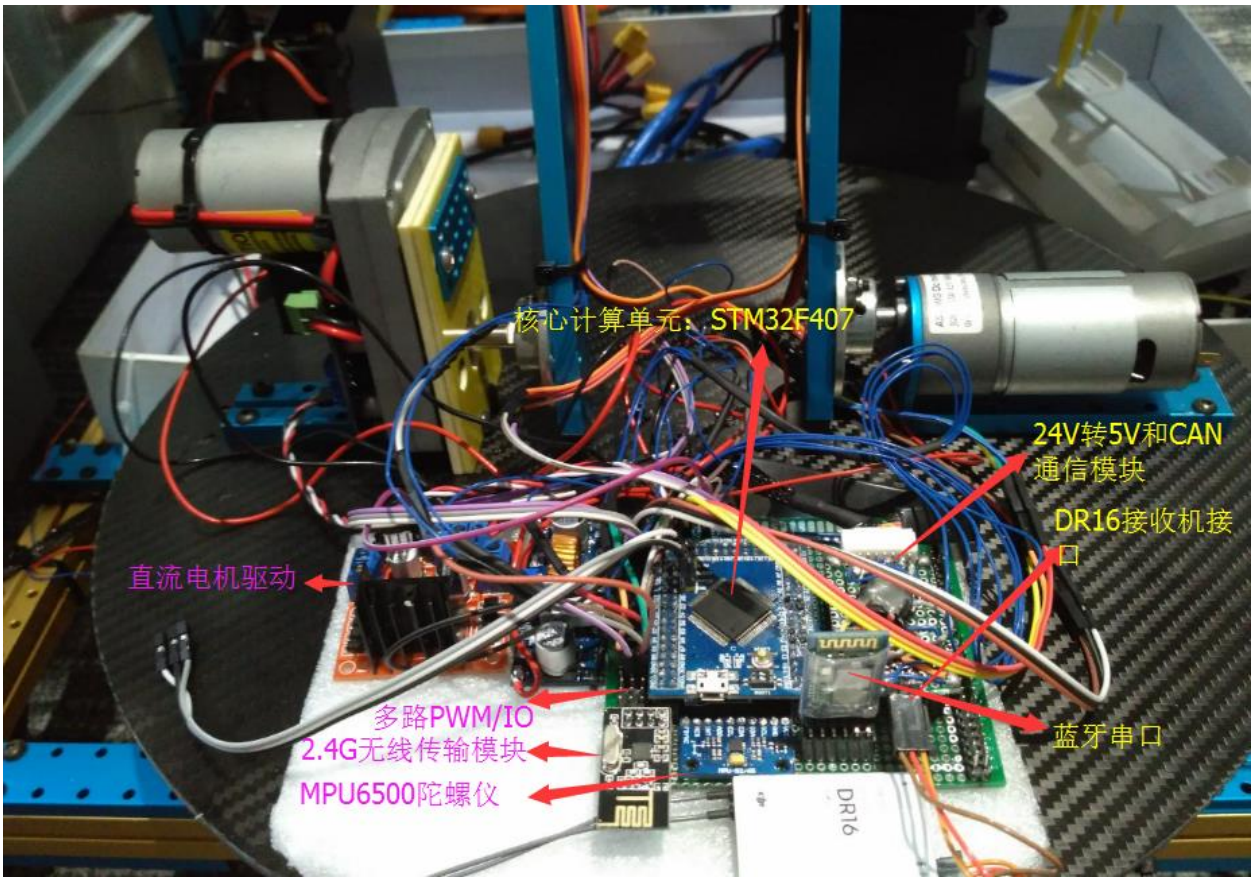
1、地面机器人移动与定位方案：采用 Robomasters 提供的 Mecanum 轮、Makeblock 部件搭建四轮全向移动平台，用 EC60 无刷编码电机驱动，根据底盘电机编码器反馈的角度值和 Mecanum 运动力学传输方程解算底盘相对任意初始时刻的姿态；也可以给定任意位姿，用 Mecanum 逆运动学解析公式推算各个轮子的转角，然后用误差控制器来使平台趋近目标位置。

2、抓取机构方案：采用 4 自由度抓取机构，第一自由度控制整个抓取机构的横摆（YAW），由 RM6025-YAW 轴云台电机驱动；第二自由度控制手臂的俯仰，由一个大扭力舵机驱动；第三自由度控制手腕转角，由两个标准舵机驱动，第四自由度控制机械爪的开合，由两个标准舵机驱动。

3、储物仓与升降台方案：makeblock+硬纸盒+直流电机+限位开关+丝杠。

核心控制板：

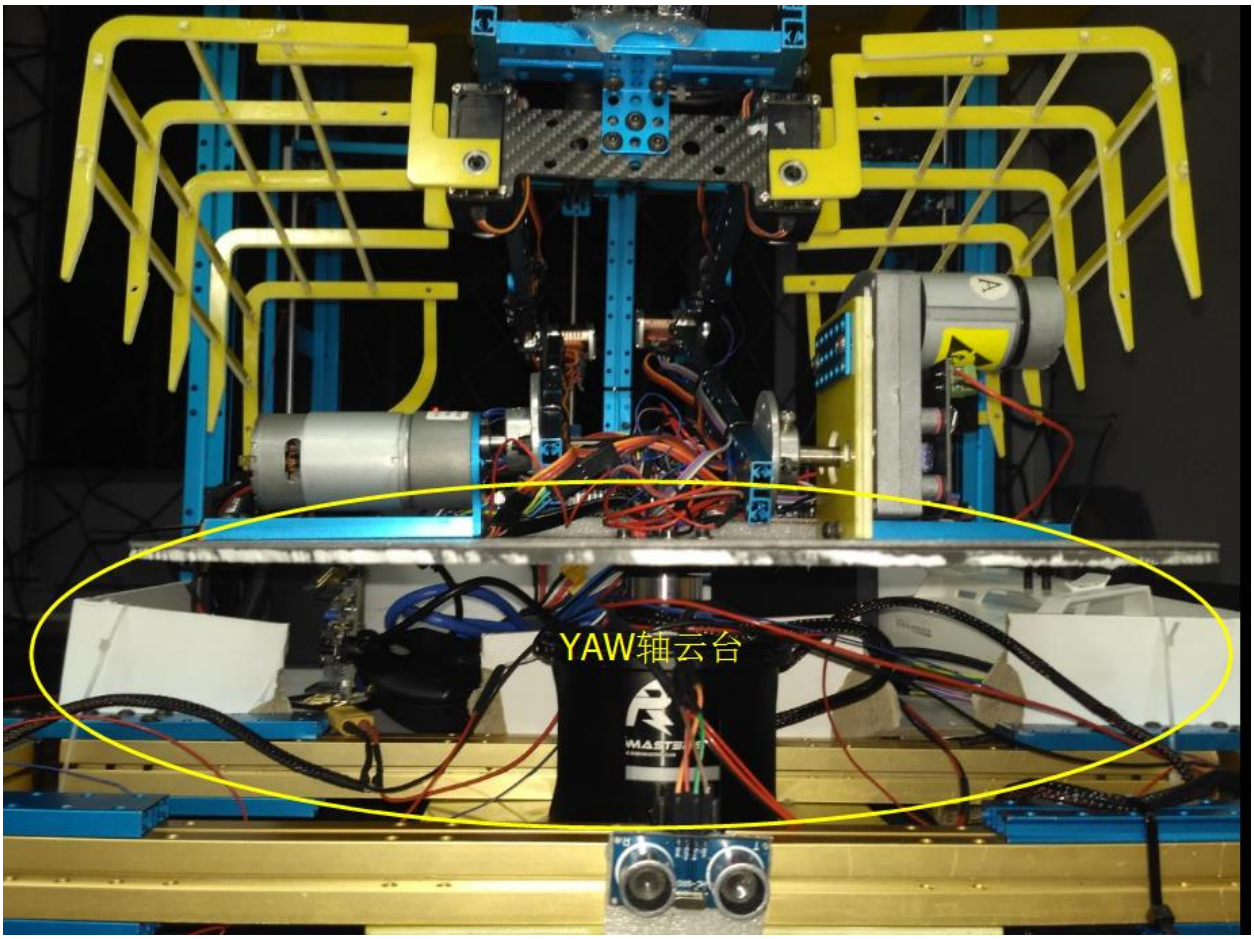
根据所需要控制的机构类型和数量，我们自己做了一个全能的主控板，代替 RM 步兵战车的梯形板，用以控制车上所有的功能单元，如下图所示：



主板为手工焊接，外加其他模块（通信、驱动电路）搭建而成，功能强悍，精巧耐用

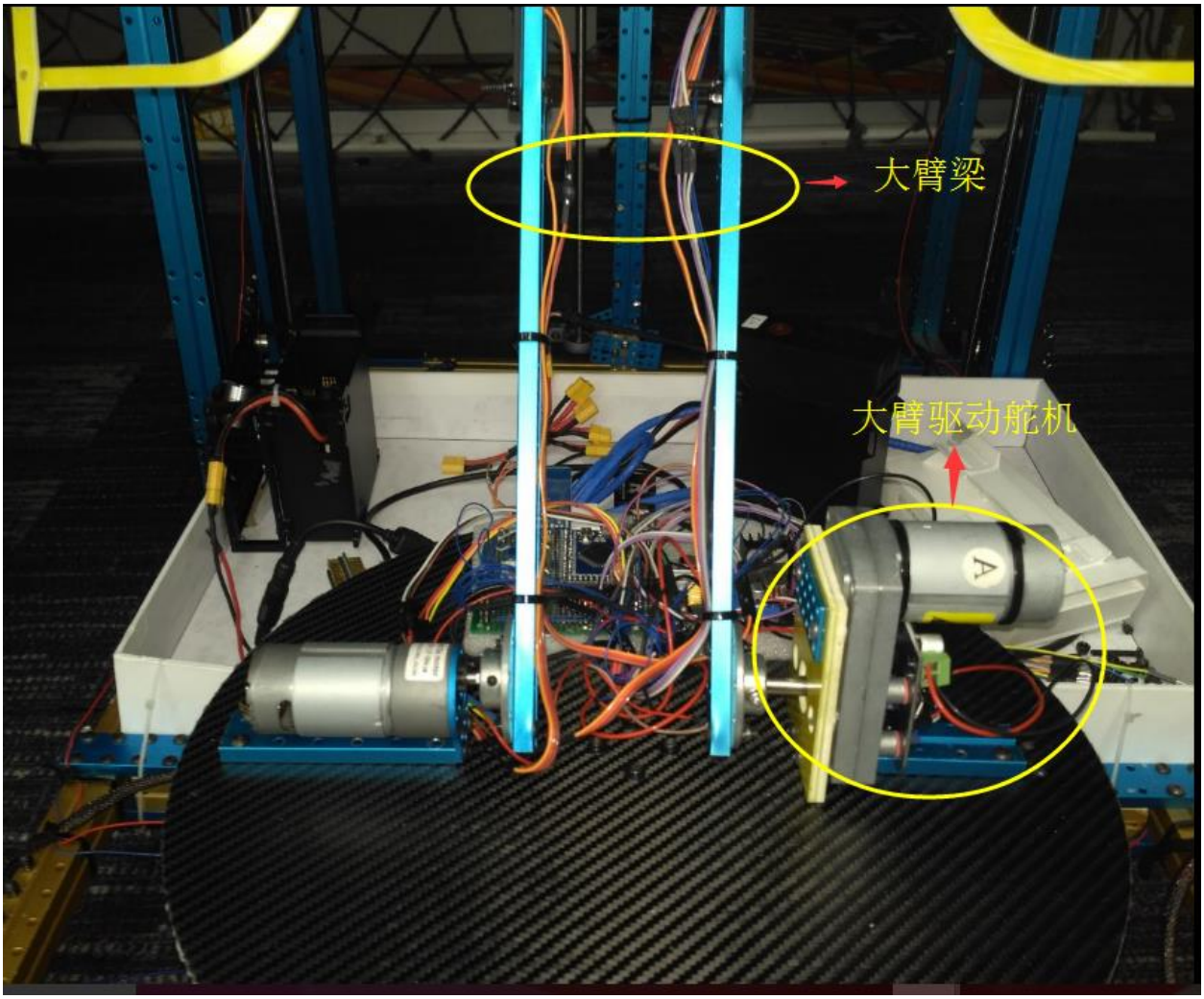
其他功能单元：

1、YAW 轴云台



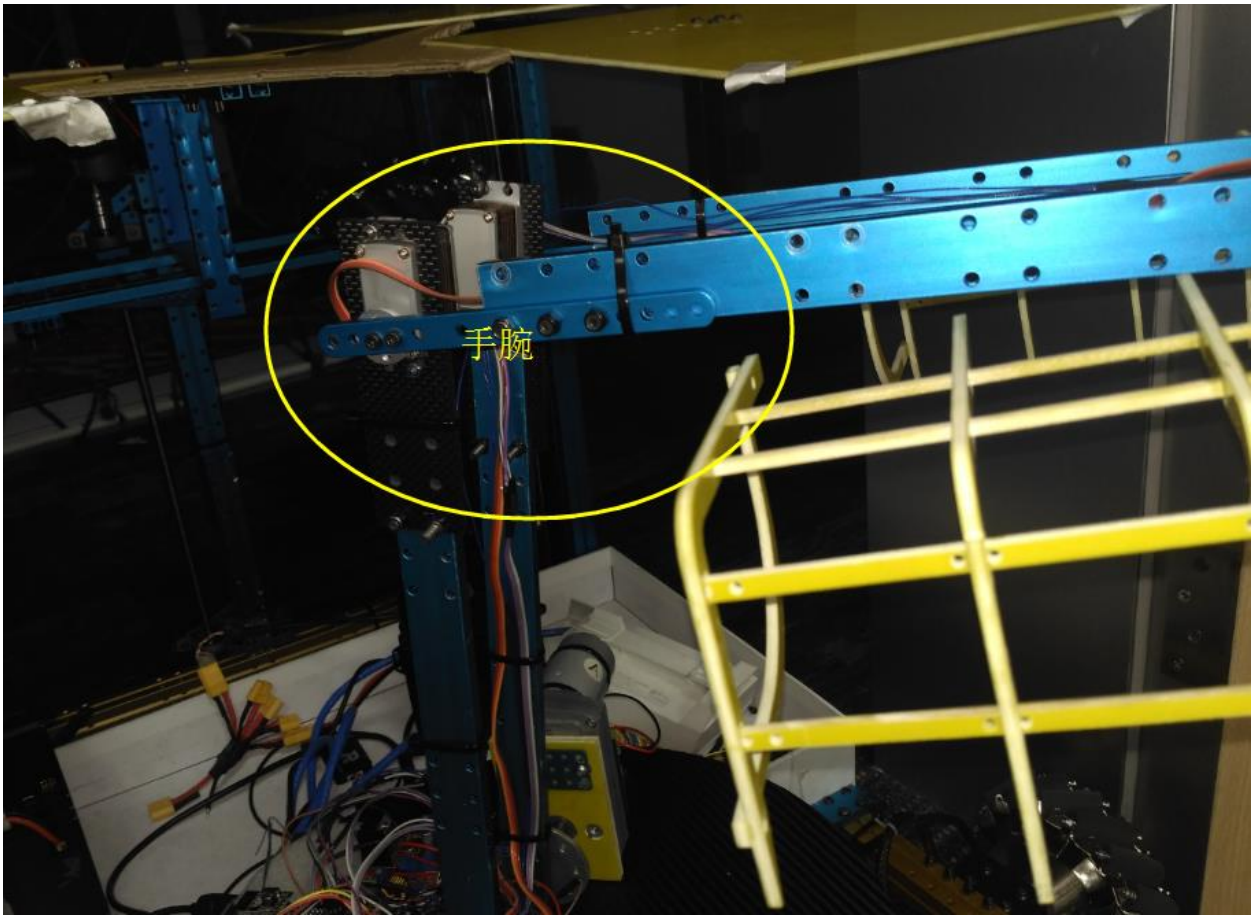
由 RM6025-YAW 编码直驱电机驱动，可为上方抓取机构提供 0-180 度的横摆角度。

2、手臂（PITCH）



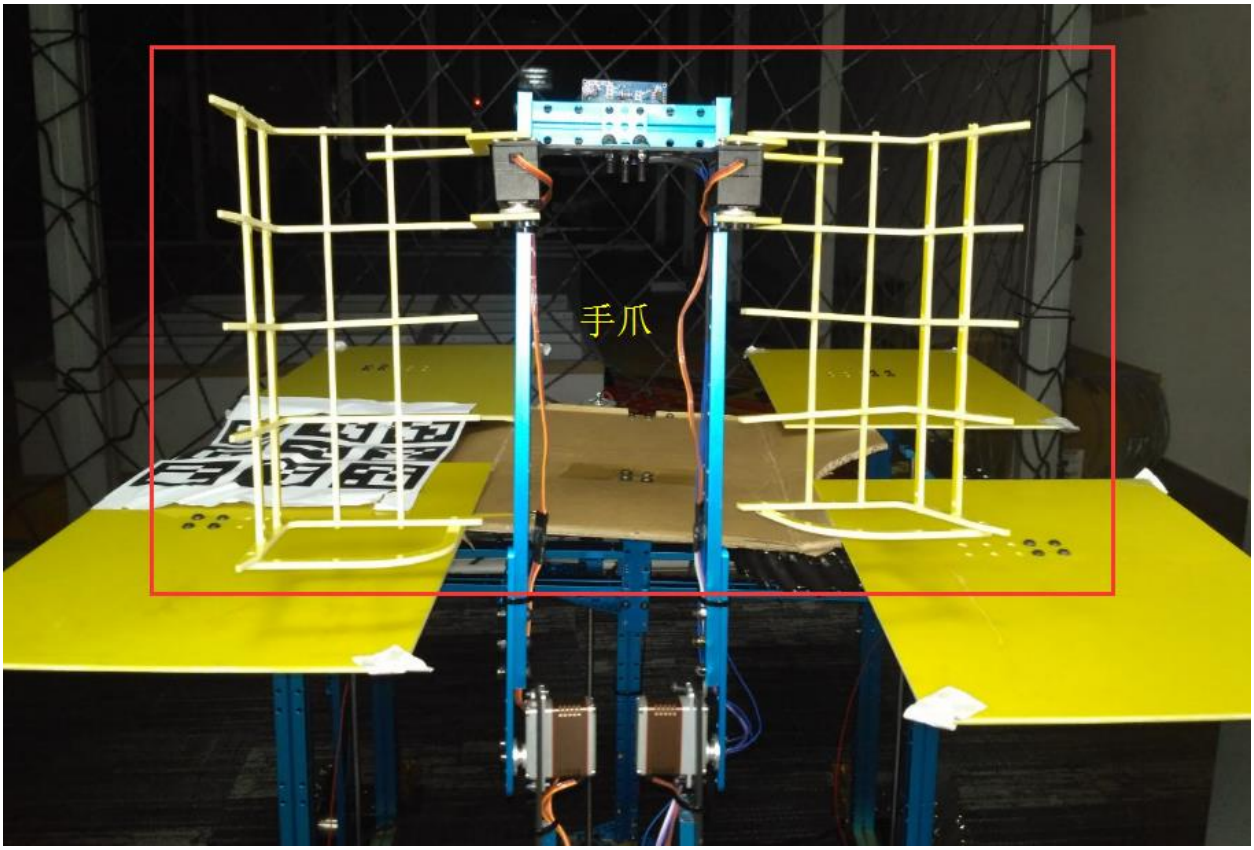
由 makeblock 铝件、一个大扭矩舵机和一个从动轴搭建而成，可为上方的抓取机构提供 0-180 的俯仰角度。

3、手腕



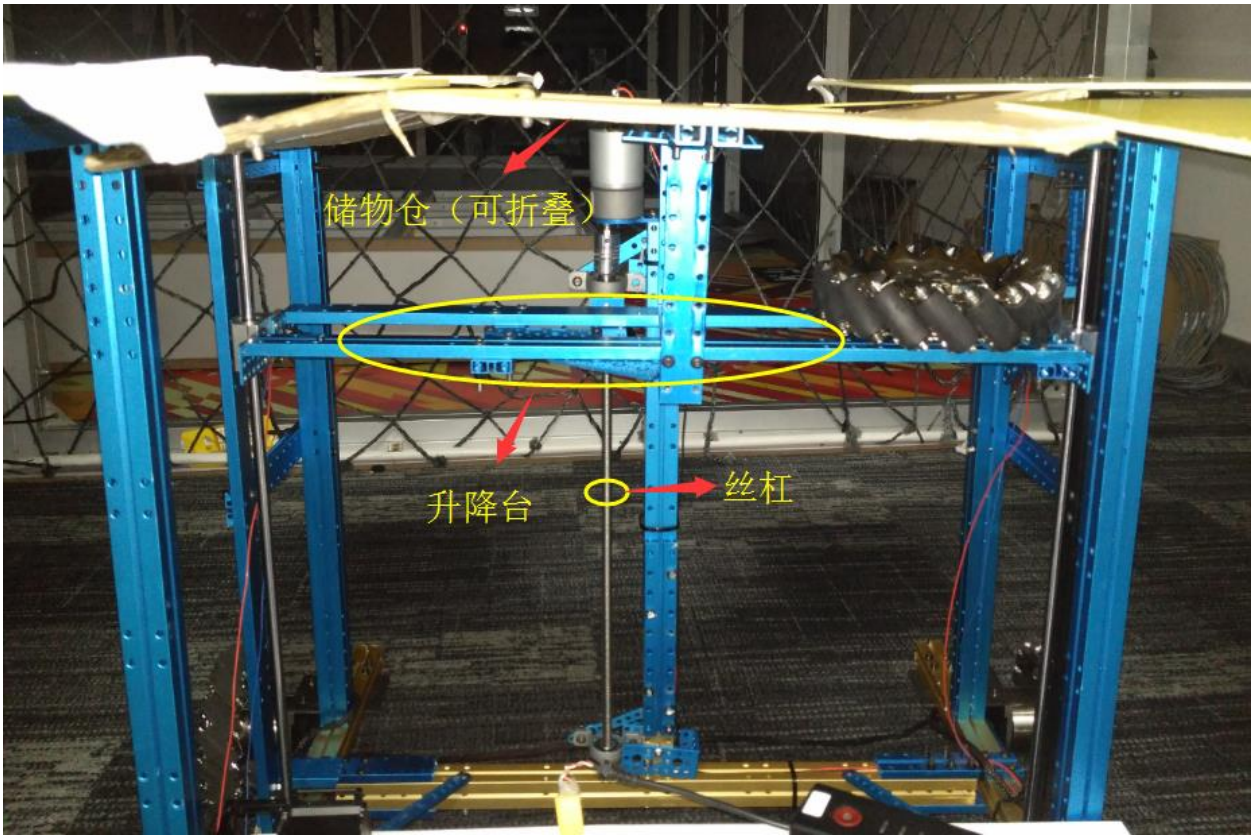
由两个标准舵机和轴承组成，可调整爪子朝向，使目标抓取更容易。

4、手爪



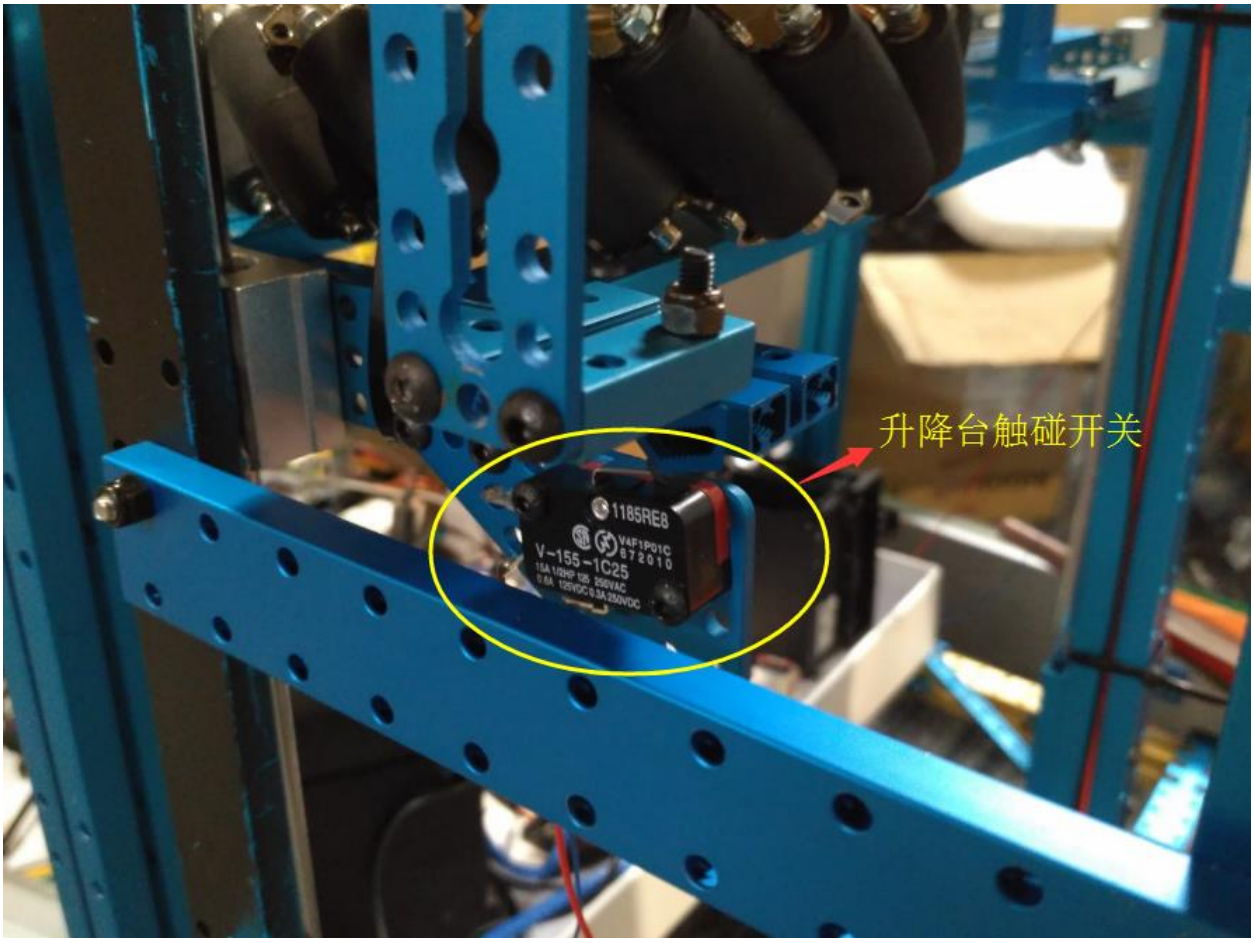
由两个标准舵机、轴承和环氧树脂构成。

5、储物仓与升降台



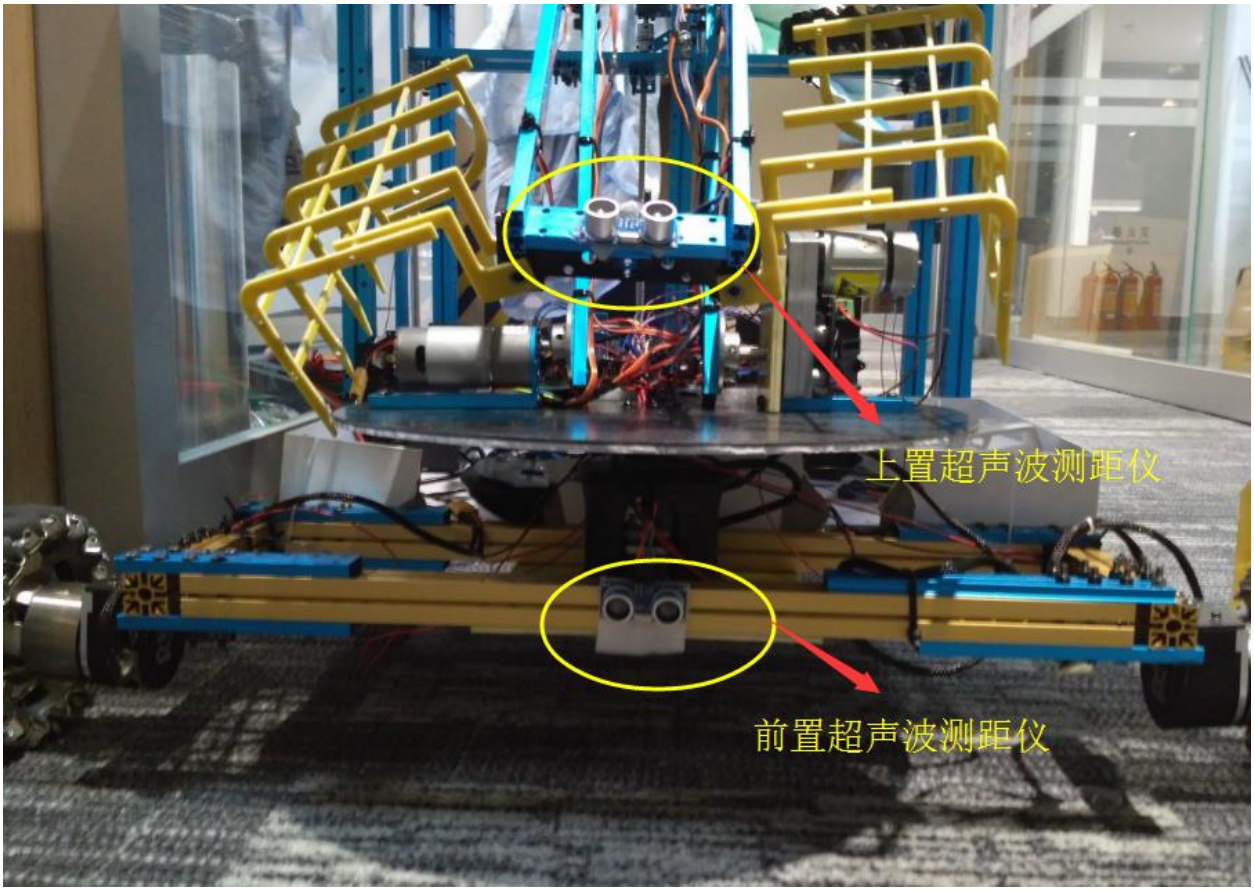
由可折叠硬纸盒、丝杠、直流电机构成，可控制储物仓的升降。

6、升降台触碰开关



用于对升降台进行电子限位。

7、超声波测距仪

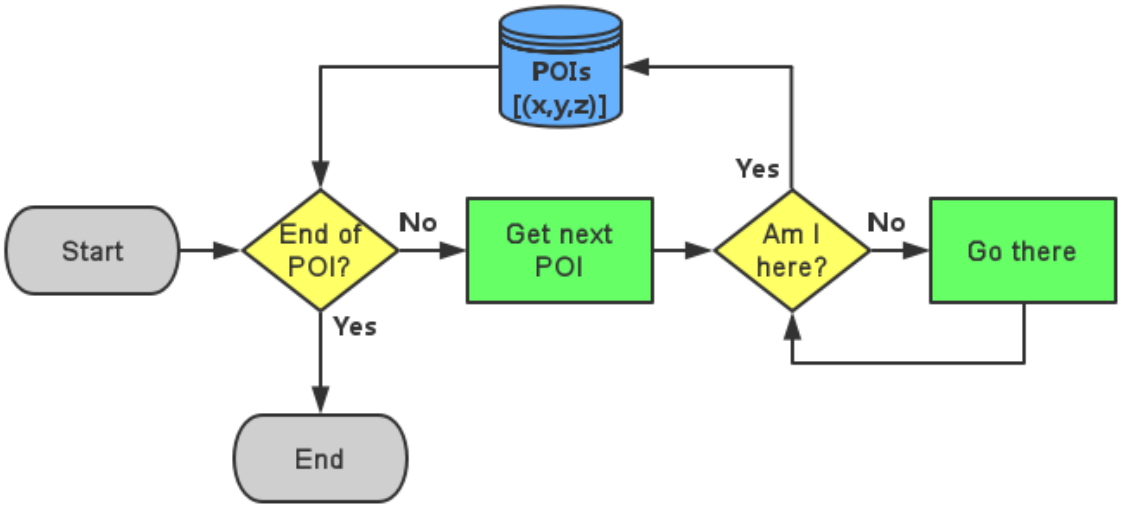


前置超声波测距仪：用于与放置公仔的盒子保持合适的距离

上置超声波测距仪：用于检测空中机器人是否降落

软件流程：

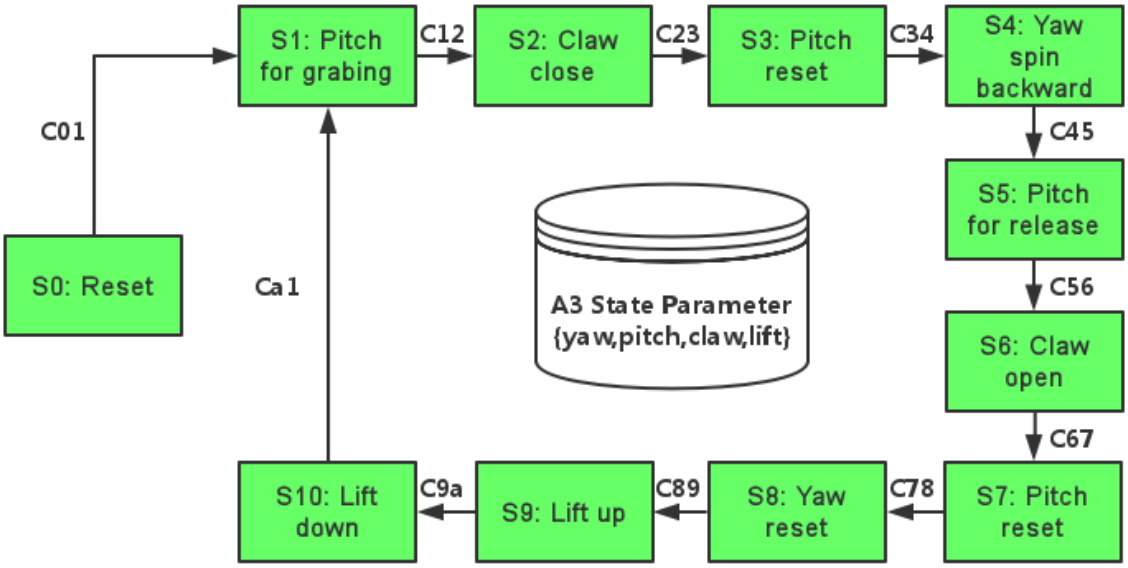
1、从 1 区到 3 区（Area1 to area3）,兴趣点模式（Points of Interest Mode）



Point of Interest Mode
(By Bangjie Mo)

说明：系统会根据场地状况预设一系列兴趣点（Points of Interest），可认为是一个位姿数组[(x, y, z)]（x, y 代表地面机器人在游戏竞技场上的坐标，z 为偏航值），从 1 区到 3 区的任务开始时，程序会顺序读取这些兴趣点，然后以一定的速度依次到达，直到最后一个兴趣点。这样做的好处是，可以增加任意多的兴趣点，让地面机器人沿着预定的轨迹走（比如沿着场地边缘走，可以避开移动篮筐，避免碰撞）。如果把地面机器人和空中机器人的位置坐标互相共享，可以实现地空互补引导/追踪。

2、3 区任务状态机（A3 State Machine）



A3 State Machine
(By Bangjie Mo)

说明：在 3 区底盘为静止状态，主要的控制对象为抓取机构和升降台，即{yaw, arm, wrist, claw, lift}；而爪子须与地面平行，即大臂和小臂的角度互补（大臂为主动，小臂为从动），故将大臂和小臂的控制变量合为一个（pitch）,状态变量变为{yaw ,pitch, claw, lift}。为简化操作流程，将 3 区的任务分解为 11 个步骤，且相邻步骤之间只有一个状态变量的值发生变化：

第 0 个步骤（S0）：复位状态（YAW 朝前，PITCH 为预设值，爪子张开，升降台位置在下方）

第 1 个步骤（S1）：PITCH 调整到适合抓取公仔时状态（准备抓娃娃）

第 2 个步骤（S2）：爪子关闭（开抓）

第 3 个步骤（S3）：PITCH 复位（抬爪）

第 4 个步骤（S4）：YAW 朝后（准备将爪子伸入储物仓）

第 5 个步骤（S5）：PITCH 调整到适合释放公仔时状态（将爪子伸入储物仓上方）

第 6 个步骤（S6）：爪子张开（将公仔放进储物仓）

第 7 个步骤（S7）：PITCH 复位（准备复位 YAW）

第 8 个步骤（S8）：YAW 复位（装备升仓）

第 9 个步骤（S9）：升仓（准备与飞机交接）

第 10 个步骤（S10）：储物仓复位（准备下一轮抓取）

状态之间的跳变可能需要满足一定的条件（Condition）,这里有 11 个状态跳变，所以可能的条件有 11 个，将他们分别命名为 C01~Ca1，他们分别代表：

状态 0 到状态 1 跳变条件（C01）：任务开始执行，所有的状态变量均复位成功

状态 1 到状态 2 跳变条件（C12）：PITCH 预备抓取的动作已完成

状态 2 到状态 3 跳变条件（C23）：爪子的抓取动作已完成

状态 3 到状态 4 跳变条件（C34）：PITCH 已复位

状态 4 到状态 5 跳变条件（C45）：YAW 已转向正后方

状态 5 到状态 6 跳变条件（C56）：PITCH 已为公仔释放做好准备，且超声波数据稳定小于某一阈值（飞机已降稳）

状态 6 到状态 7 跳变条件（C67）：爪子已张开

状态 7 到状态 8 跳变条件（C78）：PITCH 已复位

状态 8 到状态 9 跳变条件（C89）：YAW 已复位

状态 9 到状态 10 跳变条件（C9a）：升降台位于上方（上方触碰开关为闭合状态），且与飞机完成公仔的交接

状态 10 到状态 1 跳变条件（Ca1）：升降台已归位（下方触碰开关为闭合状态）

兴趣点模式关键代码：

```
/************************************/
/*      Mission 1: Go to A3         */
/*     Point of Interst Mode        */
/************************************/
/************************************/
/*            场地示意              */
/*  3区                      4区    */
/*         _________________        */
/*        |    |     |    |         */
/*        |____|     |____|         */
/*        |                |         */
/*        |                |         */
/*        |____      ____| x       */
/*        |    |     |    |         */
/*        |____|_____|____|         */
```

```
/*                    y                */
/*  2区                          1区  */
/*                                     */
/*   地面机器人头朝y正方向摆放      */
/*************************************/
/*************************************/
/*        地面机器人运动轨迹          */
/*       标有箭头的地方为兴趣点       */
/*  3区                          4区  */
/*          ________________           */
/*         |      <         |          */
/*         |      |         |          */
/*         |>----^          |          */
/*         ||               |          */
/*         |^----<          | x        */
/*         |      |         |          */
/*         |____  ^---< ____|          */
/*                    y                */
/*  2区                          1区  */
/*                                     */
/*   地面机器人头朝y正方向摆放      */
/* 按此轨迹走可避免与移动篮筐碰撞   */
/*************************************/
#define POI_NUM 7  //兴趣点个数

static const ChassisPosition POI[POI_NUM] =
{

        { 0, 0, 0},  //第0个兴趣点：
         { 0, 3, 0},  //第1个兴趣点：
         { 1, 0, 0},  //第2个兴趣点：
         { 1, 4, 0},  //第3个兴趣点：
         { 3, 4, 0},  //第4个兴趣点：
         { 3, 3, 0},  //第5个兴趣点：
         { 4, 3, 0},  //第6个兴趣点：
};

static uint8_t i = 0; //兴趣点索引

/* 判断当前兴趣点是否已到达 */
#define POI_ACU 0.01  //精度控制
static uint8_t I_AM_HERE(void)
{
         return abs(POI[i].x - chassisPositionXPid.fdb) < POI_ACU &&
                     abs(POI[i].y - chassisPositionYPid.fdb) < POI_ACU &&
                abs(POI[i].z - chassisPositionZPid.fdb) < POI_ACU;
}
```

```
/* 兴趣点有限状态机 */
void POIFSM(void)
{
        static uint8_t count = 0;  //count for confirmability
        if(i < POI_NUM)  //End of POI?
        {
                if(I_AM_HERE())  //Am I here?
                {
                        count++;
                        if(count > 50)
                        {
                                i++;
                        }
                }
                else
                {
                        count = 0;
                }
        }
}

/* 到指定兴趣点 */
void GoToPOI(void)
{
        POIFSM(gc);
        if(i < POI_NUM)
        {
                chassisPosition = POI[i];
        }
}
```

3 区任务状态机代码：

```
/*********************************************/
/*       Mission 2: Grab toys and            */
/*        pass it to aerial robot            */
/*          -Fixed Point Mode-               */
/*********************************************/

#define STATE_NUM 11  //状态数为 11

static uint8_t s = 0; //初始状态为 0

/*********************************************/
```

```
/*     Time(ms) Pause for Each Stage      */
/*****************************************/
static const uint16_t pause[STAGE_NUM] =
{
        2000,  //Waiting for state initialization
        2000,  //Waiting for pitch ready for grabing toys
        3000,  //Waiting for claw close
        2000,  //Waiting for pitch reset
        2000,  //Waiting for yaw spin backward
        2000,  //Waiting for pitch ready for releasing toys
        3000,  //Waiting for claw open
        2000,  //Waiting for pitch reset
        2000,  //Waiting for yaw reset
        5000,  //Waiting for lift up
        5000,  //Waiting for lift down
};

/* Check if yaw is backward */
uint8_t IsYawBackward(void)
{
        return (RM6OEncoder5.angle > 175 && RM6OEncoder5.angle < 185);
}

/* Check if yaw is reset */
uint8_t IsYawReset(void)
{
        return (RM6OEncoder5.angle < 5 && RM6OEncoder5.angle > -5);
}

static uint32_t t = 0;     //timer for current state
static uint8_t count = 0;  //count for comfirmability
void A3FSM(void)
{
        t++;
        if(s == 4)
        {
                if(IsYawBackward())
                {
                        count++;
                        if(count > 50)
                        {
                                s++;
                                t = 0;
                        }
                }
                else
                {
```

```
                        count = 0;
                }
        }
        if(s == 6)
        {
                if(SONAR[0].data < 20)
                {
                        count++;
                        if(count > 50)
                        {
                                count = 0;
                                s++;
                                t = 0;
                        }
                }
                else
                {
                        count = 0;
                }
        }
        if(s == 8)
        {
                if(IsYawReset())
                {
                        count++;
                        if(count > 50)
                        {
                                s++;
                                t = 0;
                        }
                }
                else
                {
                        count = 0;
                }
        }
        if(s == 9)
        {
                if(ReadKey1() == 0)
                {
                        count++;
                        if(count > 50)
                        {
                                s++;
                                t = 0;
                        }
                }
```

```
                else
                {
                        count = 0;
                }
        }
        if(s == 10)
        {
                if(ReadKey2() == 0)
                {
                        count++;
                        if(count > 50)
                        {
                                s++;
                                t = 0;
                        }
                }
                else
                {
                        count = 0;
                }
        }
        if(t > pause[s])
        {
                s++;
                t = 0;
        }
        s %= STAGE_NUM;
}

void A3Mission(void)
{
        A3FSM(void);

        // yaw

        if(s < 4)

        {
                yaw = YAW_RESET;
        }
        else
        {
                yaw = YAW_BACKWARD;
        }
        // pitch

        if(s < 1)
```

```
{
        pitch = PITCH_RESET;
}
else if(s < 3)
{
        pitch = PITCH_GRABING;
}
else if(s < 5)
{
        pitch = PITCH_RESET;
}
else if(s < 7)
{
        pitch = PITCH_RELESE;
}
else
{
        pitch = PITCH_RESET;
}
//claw

if(s < 2)

{
        claw = CLAW_OPEN;
}
else if(s < 6)
{
        claw = CLAW_CLOSE;
}
else
{
        claw = CLAW_OPEN;
}
//lift

if(s < 9)

{
        lift = LIFT_RESET;
}
else if(s < 10)
{
        if(ReadKey1() != 0)
                lift = LIFT_UP;
}
```

```
        else
        {
                if(ReadKey2() != 0)
                        lift = LIFT_RESET;
        }
}
```

总结

就嵌入式方面的任务来谈，本次 RM 夏令营对营员的的实现方案没有硬性要求（视觉或雷达或其他），但是却非常考验营员的细节处理与优化能力。比如我们在用编码电机+mecanum 轮来实现空间全向移动和定位的时候就发现，定位的结果往往不是特别精准（特别是 YAW 轴），后面我们通过陀螺仪来进行融合和校准，效果就好了很多，基本能达到方案指标。遗憾的是，由于前期人员任务分配不均衡，我们的硬件装配被拖了后腿，导致后面的软件设计也受影响，没能让地面机器人在比赛之前完全实现预计的全部功能，这也让我们在埋头苦干之后反思良久。不过让我们倍感欣慰的是，我们的算法思路和实现方式经过验证之后是可行的，从发现问题到解决问题，我们迂回往返，终于一步步靠近真理，品尝到成功的喜悦和快感。

眼看夏令营就要结束，心中除了千般不愿万般不舍，还充满了 Robomasters 的精益求精和 DJI 的创新精神。下一个夏日，我还要像现在一样，在 DJI 的熔炉下红红火火地燃烧，热热烈烈地绽放。