目录

1. 零件清单

1.1 标准件清单

1.1.1 主控板 Arduino UNO R3

1.1.2 驱动扩展板 Motor Shield V5.2

1.1.3 电源模块

1.1.4 五路循迹模块 Line Follower Array5s

1.1.5 直流减速电机 JGA37，1：43

1.1.6 麦克纳姆轮　60mm

1.1.7 多功能支架

1.1.8 长 U 标准舵机支架

1.1.9 标准舵机圆盘

1.1.10 L 形标准舵机支架

1.1.11 法兰杯士轴承

1.1.12 单轴舵机　MG996

1.1.13 单轴舵机　YF-6125

1.1.14 机械爪

1.1.15 侧板

1.2 自制零件清单

1.2.1 小车底板

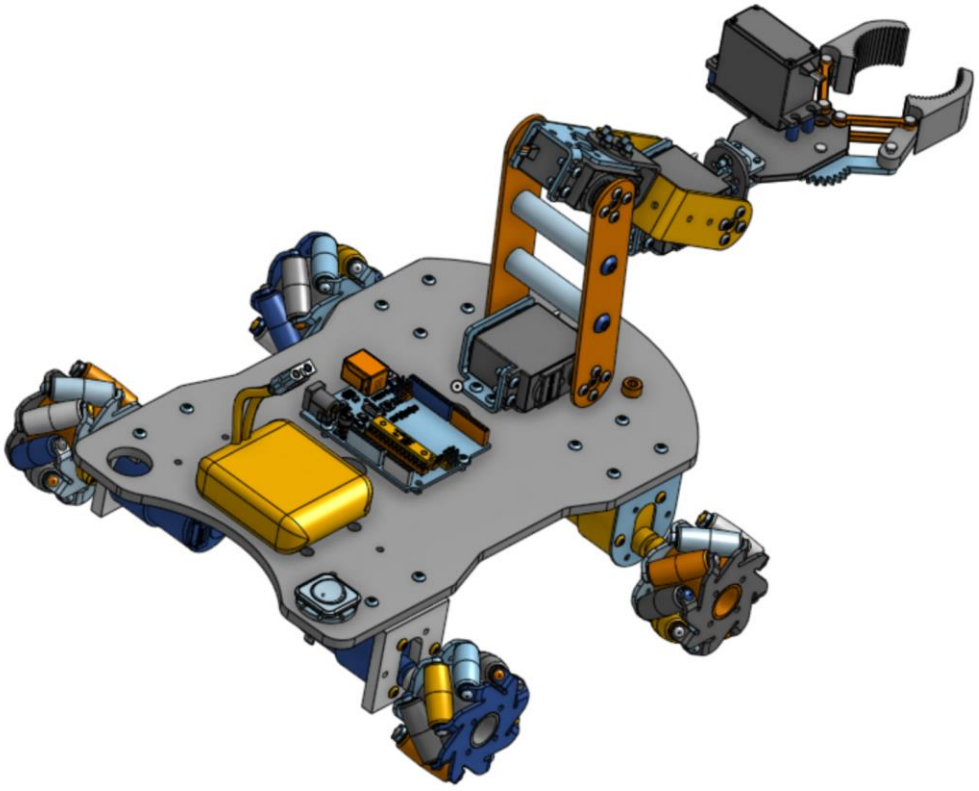
1.2.1.1 材料：木板
1.2.1.2 尺寸：265mm*170mm
1.2.1.3 加工工艺：激光切割
1.2.2 电机支架

1.2.2.1 加工工艺：铣工
1.2.3 横梁

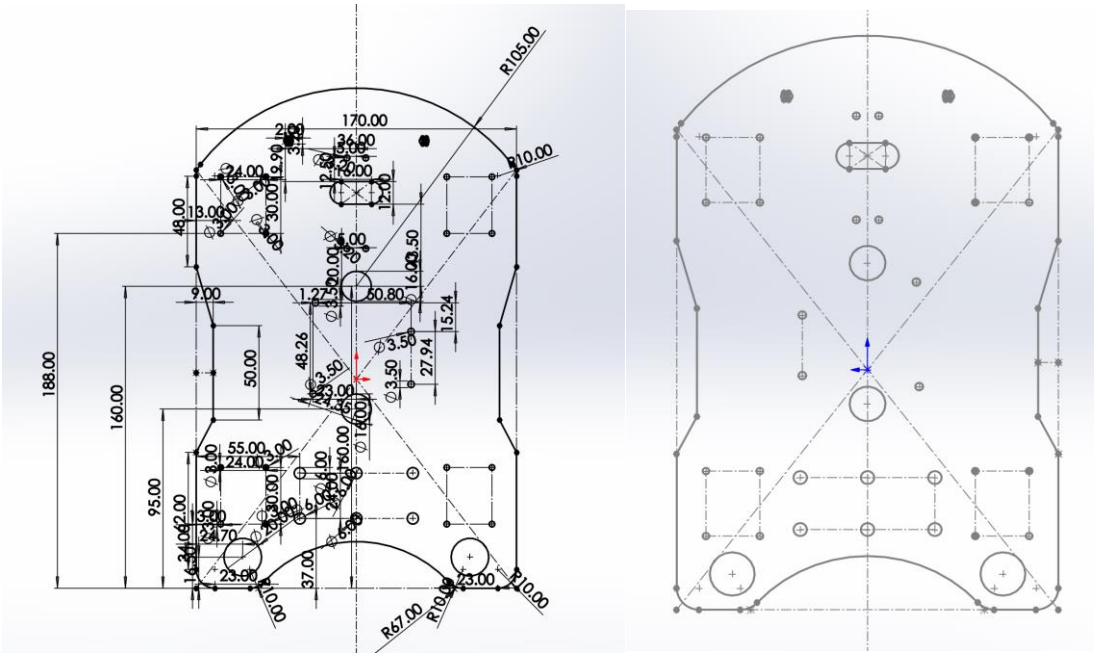
1.2.3.1 加工工艺：车工
要求写明自制零件的材料、尺寸以及对应的加工工艺，图文并茂。

2. 机械结构图

整体结构参照所给样车进行设计与装配，由于电池设备、Arduino 板的不同对底板进行了部分程度的重新设计与改进（改动集中在中部打孔部分的尺寸）。



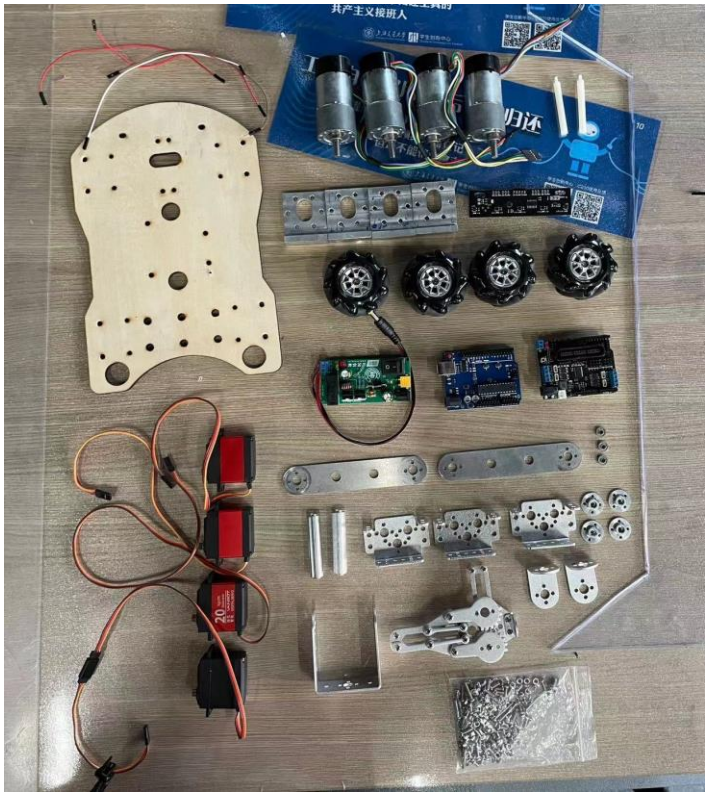
底板 SolidWorks 设计图见下



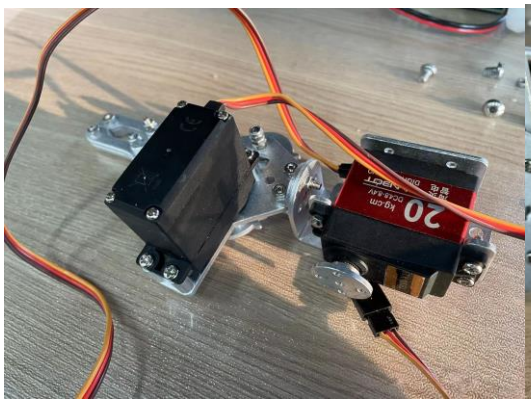
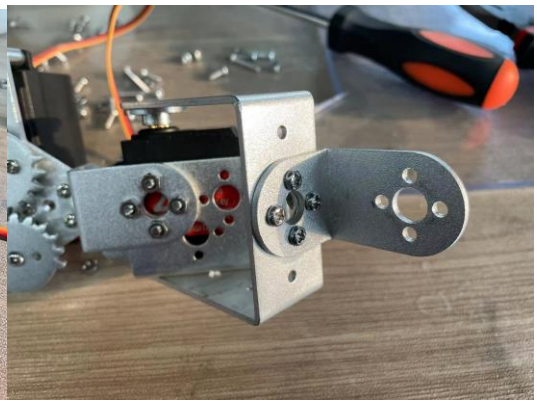
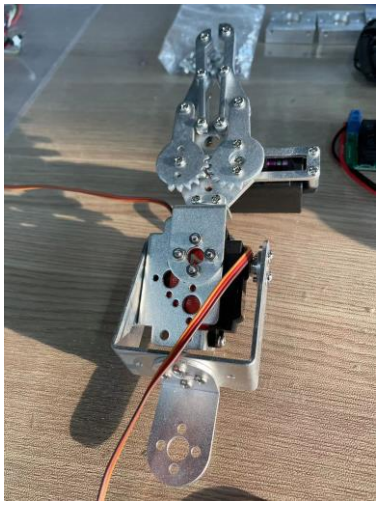
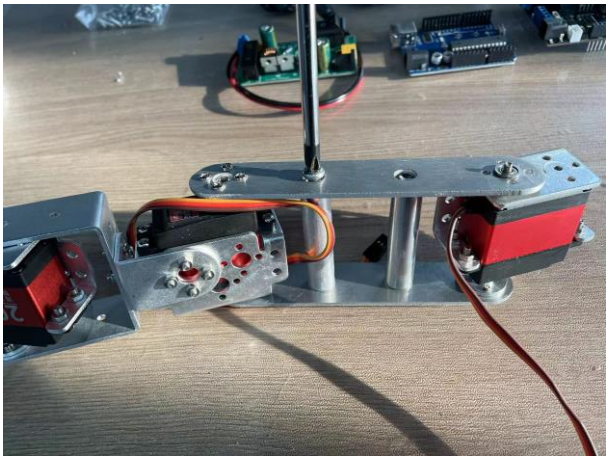
通过激光切割，得到成品如下

3. 装配步骤

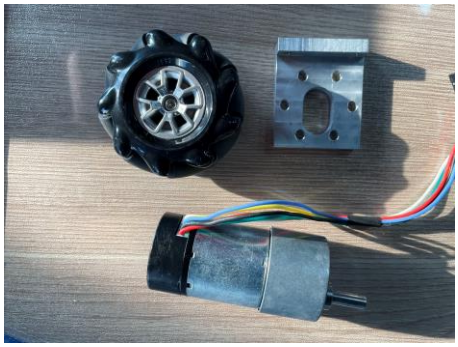
第一步，激光切割出小车底板并且准备好所有材料。



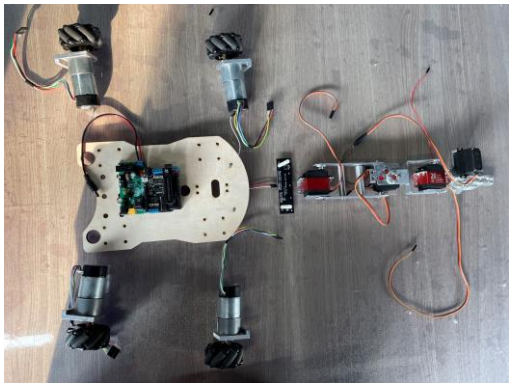
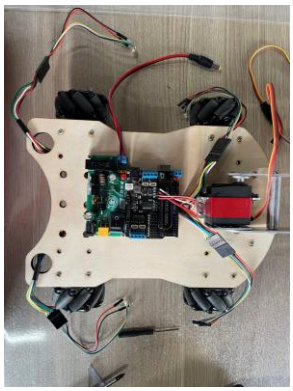
第二步，组装机械臂。将 MG996 舵机与机械爪相连，将剩余 3 个 YF-6125 舵机分别与对应的舵机支架固定好后，依次序进行连接组装，具体流程如下：

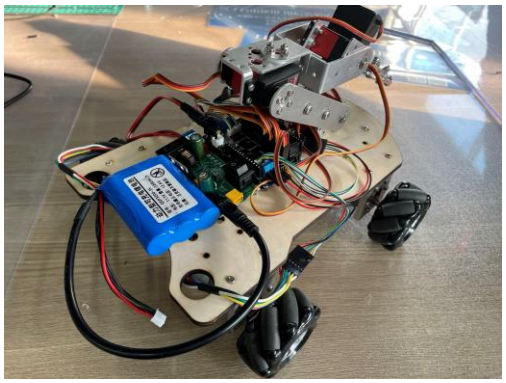
第三步，组装麦轮，将麦轮轴通过电机支架与电机连接。




第四步，预组装，将 ARDUINO 板、拓展板固定在小车底盘，将车轮、巡迹传感器与机械臂依次固定在底板上。

第五步，将各模块接线入拓展板，将电池接入，大功告成！



4. 控制系统

4.1 控制系统连接

控制系统的供电模块由电池通过供电线与稳压模块连接，稳压模块通过输电线给 uno 板以及拓展板供电，拓展板与 uno 板通过拓展板上的管教进行代码功能的传输以及电能的传输。

4 个电机通过导线与拓展板相连接，来实现对四个电机转动的控制；4 个舵机通过导线与拓展板连接，来实现对舵机转动的控制；传感器通过四根导线与拓展板连接，通过传感器的反馈来控制小车的运动。

4.2 控制系统组件清单

图 1

图 2

图 3

1.电源稳压模块，见上图 1：
负责提供电池供电的途径，产生稳定电压，可以给拓展板供电，防止各元件烧坏。

2.Arduino uno R3 主控板，见上图 2：
与拓展板连接，使得可以通过拓展板来进行各种功能的实现。

3.驱动拓展板 Motor Shield V5.2，见上图 3：
负责控制 4 个舵机，进而控制机器臂的运动；给 Arduino uno 主控板供电、并与其进行数据

交流；通过杜邦线与传感器连接，接收传感器的数据；与四个电机连接，控制电机的运动状态，进而控制小车的运动。。

4.3 程序控制流程图

1. 直线前进 MoveForward：

```
#include <Wire.h>
#include "QGPMaker_MotorShield.h"
#include "QGPMaker_Encoder.h"

QGPMaker_MotorShield AFMS = QGPMaker_MotorShield();
QGPMaker_DCMotor *Motor1 = AFMS.getMotor(1);
QGPMaker_DCMotor *Motor2 = AFMS.getMotor(2);
QGPMaker_DCMotor *Motor3 = AFMS.getMotor(3);
QGPMaker_DCMotor *Motor4 = AFMS.getMotor(4);

QGPMaker_DCMotor *DCMotor_1 = AFMS.getMotor(1);
QGPMaker_Encoder Encoder1(1);
QGPMaker_DCMotor *DCMotor_2 = AFMS.getMotor(2);
QGPMaker_Encoder Encoder2(2);
QGPMaker_DCMotor *DCMotor_3 = AFMS.getMotor(3);
QGPMaker_Encoder Encoder3(3);
QGPMaker_DCMotor *DCMotor_4 = AFMS.getMotor(4);
QGPMaker_Encoder Encoder4(4);

void setup()
{
  AFMS.begin(50);
  Serial.begin(9600);
}
```

1. 即简单的库包含、设定管脚、初始化电机。

```
void loop()
{
  int rpm1=Encoder1.getRPM();
  int rpm2=Encoder2.getRPM();
  int rpm3=Encoder3.getRPM();
  int rpm4=Encoder4.getRPM();


  DCMotor_1->run(BACKWARD);
  Motor1->setSpeed(50);
  DCMotor_2->run(BACKWARD);
  Motor2->setSpeed(50);
  DCMotor_3->run(BACKWARD);
  Motor3->setSpeed(50);
  DCMotor_4->run(BACKWARD);
  Motor4->setSpeed(50);
```

2. 主体执行函数部分，控制四个直流减速电机向相同方向以相同速度转动即可。图中“backward”是因为小车在装配过程中前后反置，考虑到减少工作量，我们没有改动小车硬件部分而是将控制程序中所有前进与后退指令反置。

2. 弯道巡线 LineFollow:

1. 首先仍是简单的库包含、设定管脚、及电机的初始化，程序图略。

```
io.pinMode(INPUT_PIN_S0, INPUT);
io.pinMode(INPUT_PIN_S1, INPUT);
io.pinMode(INPUT_PIN_S2, INPUT);
io.pinMode(INPUT_PIN_S3, INPUT);
io.pinMode(INPUT_PIN_S4, INPUT);
```

2. 将循迹传感器的五个反馈设定为输入，用于分类判断小车与地面黑线的相对位置。

```
int stat()
{
  int adval[5];
  int ret = 0;
  for (int i = 0; i < 5; i++)
  {
    adval[i] = io.digitalRead(i);
    if (adval[i] <= 0)
      ret += (0x1 << i);
  }
  return ret;
}
```

3. 读取循迹传感器的反馈数据，从右到左为 S0-S4，黑色为 1，白色为 0,将传感器接收到的抽象数据转为一个具象的五位二进制数，方便后续调节电机的转速与方向。

```
void turnright(int v)
{
  DCMotor_1->run(BACKWARD);
  Motor1->setSpeed(v);
  DCMotor_2->run(BACKWARD);
  Motor2->setSpeed(v);
  DCMotor_3->run(FORWARD);
  Motor3->setSpeed(v);
  DCMotor_4->run(FORWARD);
  Motor4->setSpeed(v);
  delay(100);

}
```

```
void turnleft(int v)
{
  DCMotor_1->run(FORWARD);
  Motor1->setSpeed(v);
  DCMotor_2->run(FORWARD);
  Motor2->setSpeed(v);
  DCMotor_3->run(BACKWARD);
  Motor3->setSpeed(v);
  DCMotor_4->run(BACKWARD);
  Motor4->setSpeed(v);

  delay(100);
}
```

```
void moveforward()
{
  DCMotor_1->run(BACKWARD);
  Motor1->setSpeed(180);
  DCMotor_2->run(BACKWARD);
  Motor2->setSpeed(180);
  DCMotor_3->run(BACKWARD);
  Motor3->setSpeed(180);
  DCMotor_4->run(BACKWARD);
  Motor4->setSpeed(180);

}
```

4. 书写功能函数，分别为左转、右转、直行。需要使用时在主体函数中直接传递参数 V 调用即可。

```
void loop()
{
  int pos=stat();
  if(pos==B00010||pos==B00001||pos==B00011){turnright(160);}
  if(pos==B10000){turnleft(160);
  }
  moveforward();
}
```

5.主体函数部分。当接收并识别转换完毕传感器传输的数据后，根据不同的情况，执行左转、右转与直线前进的指令。

3.物块的抓取 CatchAndHold

```
#include <Wire.h>
#include "QGPMaker_MotorShield.h"

QGPMaker_MotorShield AFMS = QGPMaker_MotorShield(); //创建驱动器对象

QGPMaker_Servo *Servo1 = AFMS.getServo(0); //获取3号舵机
QGPMaker_Servo *Servo2 = AFMS.getServo(3); //获取3号舵机
QGPMaker_Servo *Servo3 = AFMS.getServo(4); //获取3号舵机
QGPMaker_Servo *Servo4 = AFMS.getServo(7); //获取3号舵机

void setup() {
  Serial.begin(9600);           // set up Serial library at 9600 bps
  Serial.println("DC Motor test!");

  AFMS.begin(50);
```

1.简单的库包含、创建驱动器对象、确定舵机的管脚，以及初始化。

```
void loop() {
//状态1: 高态
Servo1->writeServo(160);
Servo2->writeServo(140);
Servo3->writeServo(0);
Servo4->writeServo(30);
delay(5000);

//状态2:低态始
Servo3->writeServo(100);
delay(1000);
Servo1->writeServo(145);
delay(300);
Servo1->writeServo(130);
delay(300);
Servo2->writeServo(90);
delay(500);
Servo2->writeServo(75);
delay(500);
Servo2->writeServo(60);
delay(3000);
```

2.开始调整舵机参数使各舵机处于不同位置。高态即初始状态，然后通过几个步骤变换至低态。使用多步骤的原因是防止单次摆动幅度过大导致舵机损伤或小车失衡。

```
//过渡态1
Servo2->writeServo(85);
Servo3->writeServo(82);
delay(500);
Servo1->writeServo(120);
//过渡态2
Servo2->writeServo(120);
Servo3->writeServo(64);
delay(500);
Servo1->writeServo(110);
```

```
//状态5：低态终
Servo2->writeServo(140);
Servo3->writeServo(48);
delay(500);
Servo1->writeServo(100);
delay(1000);

//状态6：夹物体
Servo4->writeServo(68);
delay(1000);
```

```
//状态7:高态
Servo1->writeServo(160);
Servo2->writeServo(140);
Servo3->writeServo(0);
delay(1000000);
}
```

3.仍然通过多步控制使机械臂按要求转动，稳定夹取物块后再上升至高态位置，保持数秒后即完成任务。

4.4 控制策略描述

一、直线前进项目：

1.在足够长的距离上执行四轮同速同向转动代码，观察小车是否沿直线前进，如果稍有偏离直线，则通过微调部分电机转速进行修正。

2.在修正完毕后，因为各种调试过程中出现的偶然干扰等不稳定因素，比赛前最终调试时小车有轻微偏离直线向右的趋势。考虑到尽量减少小车的改动，我们在比赛时将小车平行放置在起跑线稍微靠左部分，最终圆满完成直线前进任务。

二、弯道巡线项目：

1.在观察赛道具体的弯道直道与转向需求后，我们注意到赛道对小车右转的要求更高。对此，因为弯道前进精度控制越高，相应的速度需要补偿性降低，我们决定忽略循迹传感器左侧第二传感器的数据，即只有当最左侧传感单元检测到黑线时才进行左转，而小车的右转则发生在最右侧传感模块检测到黑线或右侧第二传感模块或二者都检测到黑线时。这样保证了右转的精度与稳定度，而要求不高的左转则牺牲精度换取了速度。实践表明这样的思路取得了很好的效果。

2.我们采用了主函数与功能函数分离书写的方式，在需要转向或者前进时只需要在主函数中调用对应函数并传输需要的速度参数即可。这样的设置使小车的调试环节简单了很多：调试时不需要因为一个参数的变化就调整整篇代码中所有出现的相关量，而只需要在功能函数中做出对应调整即可。这样的设计让程序结构更加清晰明了，使用也更加方便快捷。

3.转弯功能函数后部的 delay()语句与检测转弯语句属于对立统一的矛盾关系。Delay 时间越长，小车执行转弯语句持续时间增大，而这段时间小车会忽略来自传感器的数据，不继续执行第二次转弯或是直行的语句。如果弯道很陡，提高 delay 值能使单次转弯效果增强，但是会降低有效检测次数即减少转弯次数；如果降低 delay 值，则单词转弯效果减弱，但是会增加有效检测次数与转弯次数。Delay 的值需要经过多次 trial and fail 后才能得到一个较为理想的值，否则小车会因为转弯次数不够或者转弯幅度过小而冲出赛道，致使循迹任务失败。

4.设计程序过程中我们充分考虑了电池电量对小车运行的影响。首次调试过程中，在循迹项目测试环节后期，小车出现了只直行不转弯的状况。首次遇到该情况的我们在反复

检查代码与小车硬件配置、各杜邦线端口连接等等可能的影响因素后仍然没有解决问题，最终我们意识到小车在直线前进与转弯两个步骤的交界处，可能是电池的电量不足以驱动瞬时的电机转动方向与速度变化，从而导致宏观上小车出现不转弯的情况。吸取教训后，我们将直线前进速度与弯道速度合理控制，避免电池电量消耗过快，也提高了程序运行的稳定性。

5. 在调试过程中我们在经历各种各样的情况后也思考出了很多应对方法，包括有些小组采用的，小车完全出线后的回转措施。对于此类情况，我们在经过考虑后决定不做此类方向上的调整，而是尽力保证地面黑线处于小车五个传感器模块所能检测到的范围内，这样可以最小化小车的行进路程，从而变相地节约全程时间。

6. 比赛前的最终调试环节中，我们设定了两套巡线方案，第一次我们采用速度较慢而稳定度很高的方案，保证拿到完赛的分数与保底时间分；第二次我们大幅提高了小车的过弯速度参数与直行速度参数，使全程时间减小到了方案一的一半，但是相应的，方案二的稳定度下降，在最终调试中只有大约 0.7 的概率能成功，而且很不幸，在最终比赛中方案二没有成功。

三、物块抓取项目：

1.宏观层面来说，我们在书写、设计程序时追求程序运行的稳定，即机械臂完成任务的稳定性，毕竟此项目对于时间没有作要求，我们在时间成本充裕的情况下应当追求其对立面的品质。从这方面来看，首先需要考虑的时物块摆放位置对任务成功与否造成的影响。对此，我们在机械臂处于低态位置时，在保证机械爪张开的同时通过数段代码块实现了机械臂的近似水平前伸，而这段前伸的范围即比赛时物块摆放的容错范围。这样的设计大大提高了任务的成功率。

2.对于每个舵机而言，我们充分考虑了其自身性质。在调试过程中，我们观察到舵机的内置转动固定速度较快，则在实际运动过程中，为了不对舵机造成太大负荷，避免造成损伤，以及防止小车因为惯性失衡或者移动，所有较大角度的舵机转动我们都采用了分段式控制执行方法，即在初态与目标终态之间加入若干过渡态，以保证舵机转动的安全性。