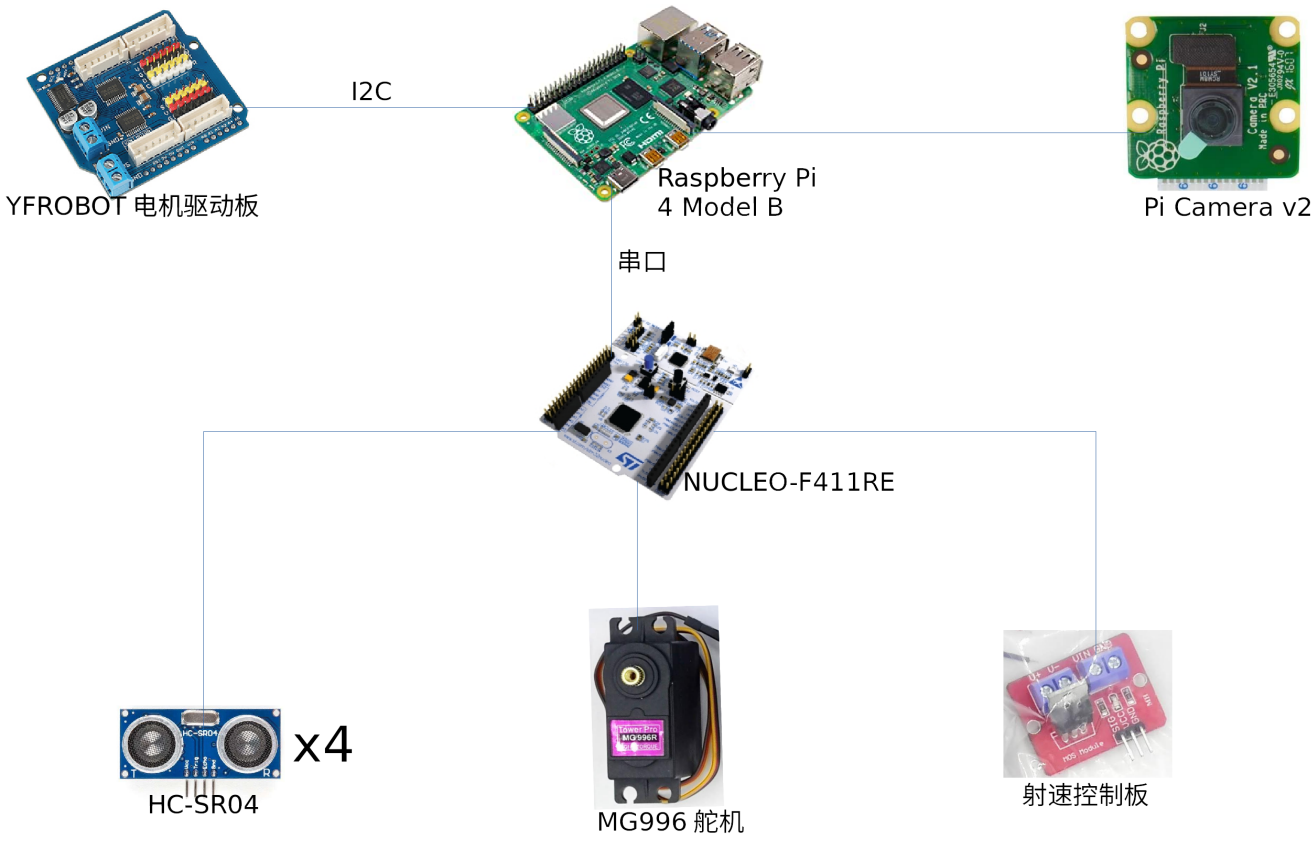
南京工业大学 RM 校内赛
迷宫射击 STAS 战队技术报告

- 作者：温颢玮 <yushijinhun@gmail.com>

概览



使用到的元件：

- Raspberry Pi 4 Model B
- Pi Camera v2
- NUCLEO-F411RE
- YFRROBOT 电机驱动板
- HC-SR04 超声测距模块 x4
- MG996 舵机
- 射速控制板
- 麦克纳姆轮小车底盘
- 水弹波箱

所有代码均已在 GitHub 上开源： https://github.com/NJTUSTAS/robomaster-2020 。代码使用 MIT 许可证。

我们使用树莓派 4B 作为主控板，其主要功能有视觉识别、接收遥控指令（通过 WiFi）、与电机驱动板及 F411RE 通信。视觉识别程序用 C++ 编写，基于 OpenCV；主控程序使用 JavaScript 编写，基于 Node。

YFRROBOT 电机驱动板通过 I2C 与树莓派通信，可以分别控制四个麦轮。

F411RE 通过串口与树莓派通信，其功能是控制舵机、射击和超声波测距模块。单片机程序基于 Mbed 开发。

视觉识别与靶标瞄准

我们实现了靶标的自动识别及瞄准功能，方法是通过摄像头抓取图像，识别出靶标中心，然后控制舵机来调整水弹枪的俯仰角，控制车轮转向来调整水弹枪的偏航角，最后使靶标中心落在视野中的预定位置。

视觉识别方面，我们使用了 AprilRobotics/apriltag 类库。我们对其提供的 `opencv_demo` 例程进行了修改，使其以机器可读的方式输出识别结果，通过管道和我们的 Node.js 主控程序进行通信。

由于树莓派算力较弱，我们在识别速度和识别精度上进行了权衡，最终选定 1024x768 作为摄像头分辨率。在此配置下，识别速度大约为每秒 4 帧。

瞄准控制方面，我们使用了两组 PD 算法，分别用于垂直方向的瞄准和水平方向的瞄准。垂直方向上，控制量为舵机的角度（即 PWM 信号占空比）；水平方向上，控制量为小车的旋转速度。

在进行水平瞄准时，由于视觉识别速度较慢（仅每秒 4 帧），我们难以获得及时的反馈，小车就很容易出现转向"转过头"的情况，即使调大 Kd 参数也无法很好地解决这一问题。为此，我们将每次转向的持续时长设为 40ms，这样转过的角度便会很小，不会因过度转向而出现振荡。

当靶标中心在视野中与预定位置的误差足够小时（垂直误差 < 3%，水平误差 < 1%），算法将不会输出控制量。然后程序将等待 500ms，若在此期间误差始终在上述范围内，我们便认为水弹枪已稳定，瞄准完成，可以进行射击。

相关代码位于 `controller/shot_target.js` 文件中。

I2C 通信

我们使用 i2c-bus 这个 Node.js 类库进行 I2C 通信。由于 YFRROBOT 电机驱动板只有 Arduino 驱动，没有 Datasheet 以及其他资料，我们花费了一些时间，将其 Arduino 驱动移植到了 Node.js 上。

相关代码位于 `controller/motor.js` 文件中。

串口通信

树莓派与 F411RE 间通过串口进行通信。我们使用的串口 Node.js 类库为 serialport。

树莓派串口配置

使用树莓派串口进行通信有两个注意点：

1. 树莓派有两个 UART，分别为 mini UART 和 PL011。
   - mini UART 功能不全，且其时钟和 CPU 核心时钟相关，因此使用时必须固定核心频率。
   - 如果要使用 PL011 则必须禁用蓝牙功能。
   - 为了使用 PL011 UART，需要编辑 `/boot/firmware/config.txt`，添加

     ```
     dtoverlay = disable-bt
     ```

2. 树莓派可能会在串口上运行 getty，需要编辑 `/boot/firmware/cmdline.txt` 禁用串口 TTY。

参考：

- https://www.raspberrypi.org/documentation/configuration/uart.md
- https://di-marco.net/blog/it/2020-06-06-paspberry_pi_3_4_and_0_w_serial_port_usage/#option-1--using-the-real-pl011-uart-port

串口通信数据格式

树莓派与 F411RE 向对方发送数据时均使用如下格式：

```
0               3              4              6
+---------------+--------------+--------------+
| leading zeros | message type | message body |
| (3 bytes)     | (uint8)      | (uint16_be)  |
+---------------+--------------+--------------+
```

我们在每个消息首部添加了 3 个零字节，以简单地处理串口通信可能出现的丢字节与字节重复错误。如此，单个错误至多影响两条消息，并且在不断发送消息的情况下，错误造成的影响十分有限。

由树莓派发送的消息：

指令含义	message type	message body
设置俯仰角	0x76 (v)	舵机 PWM 占空比 (0~65535)
射击控制	0x73 (s)	0 向射速控制板输出低电平，1 向射速控制板输出高电平
设置超声采样率	0x74 (t)	两次采样间隔（毫秒）
设置启用的超声模块	0x54 (T)	前侧 `0x1`，左侧 `0x2`，右侧 `0x4`，后侧 `0x8`；要启用多个单元，则将各单元的掩码作 OR 或运算

由 F411RE 发送的消息：

指令含义	message type	message body
前侧超声测量结果	0x66 (f)	距离（毫米）；-1 代表无穷远；-2 代表模块未连接
左侧超声测量结果	0x6c (l)	同上
右侧超声测量结果	0x72 (r)	同上
后侧超声测量结果	0x62 (b)	同上

> 由于我们原打算使用另一块电机驱动板（PWM 输入，非 I2C），因此代码中遗留了这部分控制代码。

相关代码位于 `controller/vehicle.js`、`stm32/main.cpp` 文件中。

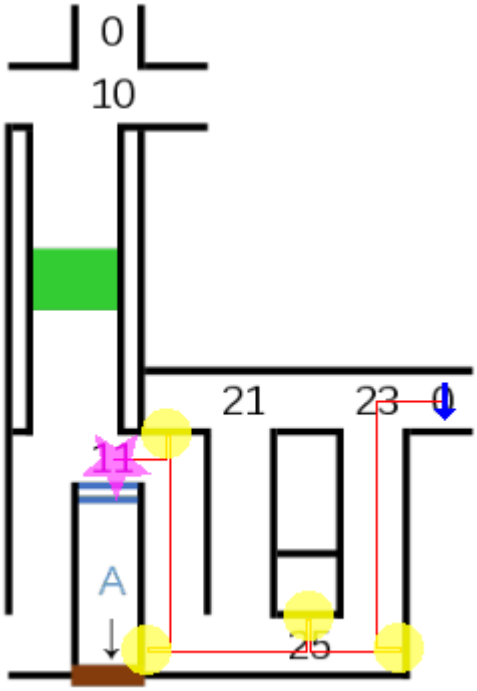
基于超声测距的自动行驶（未在比赛中使用）

我们原打算在比赛中使用基于超声测距的自动行驶，但很可惜因为预赛赛场上超声测距并没有正常工作，因此我们在决赛中被迫改用遥控。但我依然介绍一下我们在这方面所做的工作。

按照比赛规则，A、B、C 三组靶标击靶顺序是随机的，那么一共就有 6 种可能的路线。这一数字并不大，我们完全可以对行驶路线进行预编程。

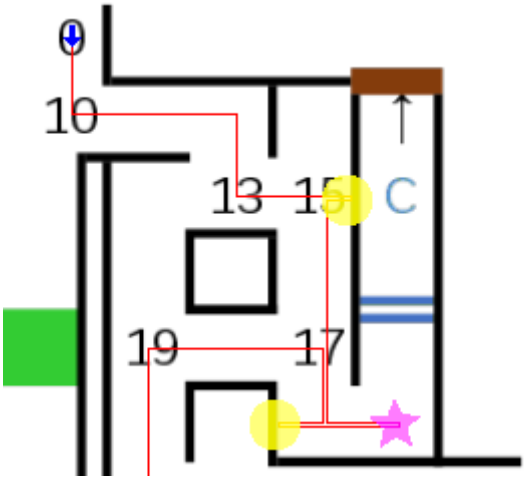
`controller/walk.js` 的 `scene2()` 函数 中包含了我们预赛行驶路线的代码，这一方案在测试场地已经通过测试。我们的自动控制主要基于以下两个原则：

- 使用超声测距作为小车移动的条件。
- 通过向左、向右、向后撞墙的方式，纠正小车的方向。



上图为预赛的行驶路线示意图。在路线中我们通过 4 次碰撞墙壁实现方向修正，这一措施在测试场地中是非常有效的。

下面是我们在另一个测试区域中的行驶路线：



测试视频： https://www.bilibili.com/video/BV1RD4y1X7Rz

遥控

我们采用的遥控方案非常简单。我们在笔记本上开启 WiFi 热点，让树莓派连接，然后通过 SSH 登入树莓派，运行遥控程序。

相关代码位于 `controller/console_control.js`。

存在的问题及改进空间

- 在实际比赛场地上，我们的超声波测距返回了异常的数据，这是由什么原因造成的？是否可以通过软件上的修改加以解决？
- 我们使用的电机带有编码器，实际上我们可以通过编码器返回的结果来精确地控制小车行进，这一点我们并未实现。
- 我们并未识别地上的 AprilTag，我们可以利用这些 AprilTag 做什么？