

从 0 开始配置自瞄代码环境(不难，也就几个小时能配好的，部分截取 coding2022 版从 0 配置)

工具（后续会整合）：

Vscode1.71.2

Foxglov-studio-1.65.0(有库需要配置，详细看官网 ros2 部分)

Linuxqq

linuxSDKV2.1.0.31（迈德相机的配置，readme 很详细，结合 coding 配置）（还有海康的，会提供相应代码启动，亲自尝试）

pigcha（科学上网）

cutecom(查看串口)

相机标定代码（camera_caliberation）

配置环境：

C++环境配置（或者用 clangd，参考 coding）

Cmake（重点，需要长时间学习）

Eigen3

Ceres

Fmt(以上几个参考 coding，fmt 需要注意静态链接安装，cmake -DCMAKE_POSITION_INDEPENDENT_CODE=TRUE ... 看 readme，不然后续使用会报错)

Opencv-4.5.5（找博客，很详细，需要花时间）

（以上部分的环境配置好后，包括相机，可以使用 2022 版自瞄进行测试）

Ros-humble(需要换源，不然下载很慢)

ros-humble-serial-driver（问 GPT 详细配置）

ros2 nav2（不确定是否需要）

gazebo

camera_info_manager（代码使用时报无这个 hpp 文件的错误，问 GPT 解决，下载后重开 vscode 或者重启解决）

还有很多库忘了（包括可视化工具的使用，看官网，下面有链接），后续会进行修改，有新的 nuc 会让学弟亲自尝试，直到代码正常使用，启动过程中会肯定会有问题的，有问题就问。

Linux 遇到的问题如下（很多博客有解决办法，一个不行多试几个）：

中文输入法

Deb 的安装，使用 dpkg

清华源

Vim 的使用

等

代码使用注意事项

相机标定

目标：确定识别的目标距离准确度

过程：

(1) 相机标定获取相机参数，方式可用 ros 或者 matlab

Ros: 通过发布节点，启动代码对节点传输的图片进行标定

code: camera_caliberation

文档: coding 相机标定

≡ 安装依赖

纯文本 ▾

```
1 sudo apt install ros-humble-camera-calibration-parsers
```

纯文本 ▾

```
1 sudo apt install ros-humble-camera-info-manager
```

纯文本 ▾

```
1 sudo apt install ros-humble-launch-testing-ament-cmake
```

纯文本 ▾

```
1 git clone https://github.com/ros-perception/image_pipeline.git
```

迈德节点: `ros2 run mindvision_camera mindvision_camera_node`

启动代码标定: `ros2 run camera_calibration cameracalibrator --size 8x6 --square 0.20 image:=/image_raw camera:=/camera` 其中 8 和 6 分别的长宽黑白格数量减 1 (即格子之间的点)

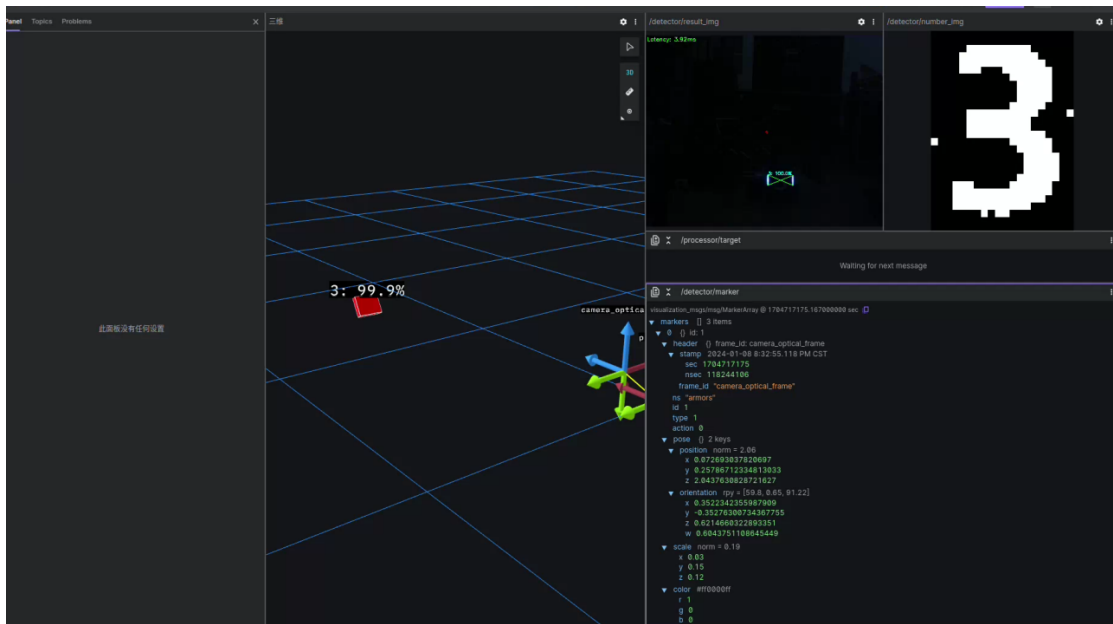
标定完后会获取一系列相机参数, 在 `ros2_mindvision_camera` 中的 `camera_info.yaml` 进行修改。

最后进行测距, 代码启动:

`ros2 run mindvision_camera mindvision_camera_node`

`ros2 launch rm_vision_bringup no_hardware.launch.py`

`ros2 launch rosbridge_server rosbridge_websocket_launch.xml`(Foxglove Studio, 有两种方法启动, 链接 <https://docs.foxglove.dev/docs/connecting-to-data/frameworks/ros2/>)



通过相机测距与真实值进行对比，保证数据准确。

(2) matlab:暂无

串口

目标：串口数据接收正确

过程：

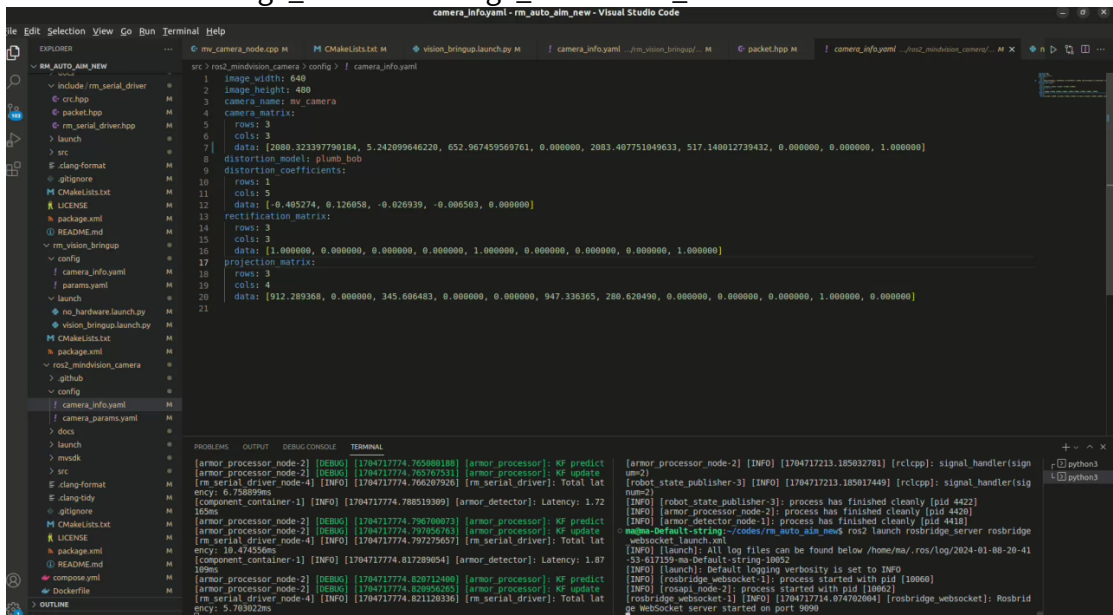
连接串口，蓝光表示已经接收数据，绿光表示已经发送数据。

首先查看是否存在串口（接了串口后）（ls /dev/tty*），查看 ttyACM0 或者 ttyUSB0，存在后记得赋权（sudo chmod 777 /dev/tty*），使用 cutecom 进行查看是否接收正确。

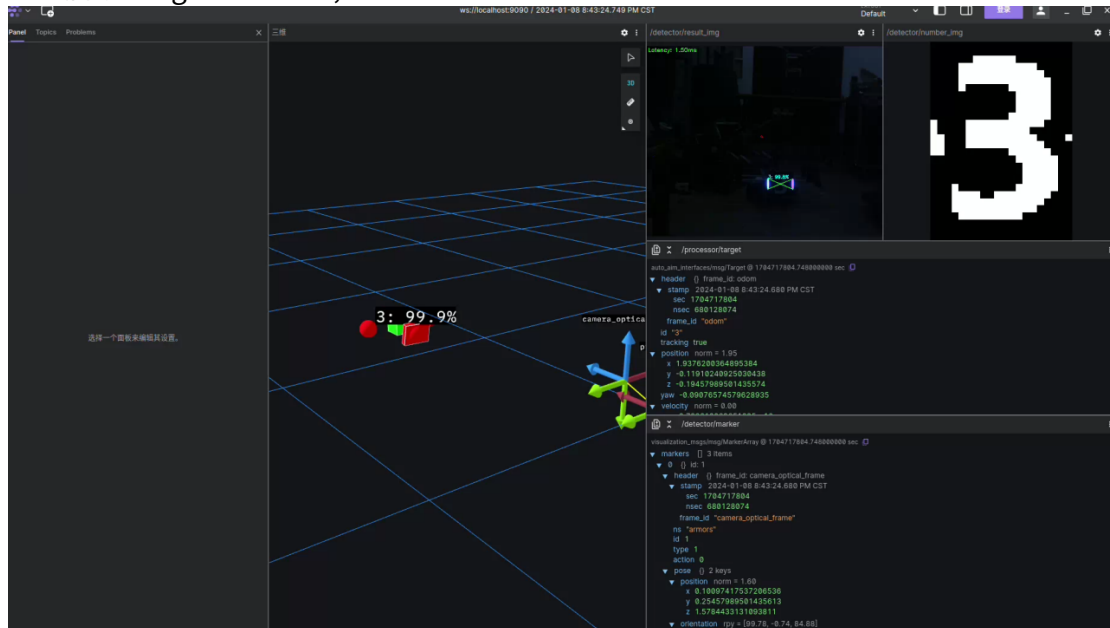
正确接收并且换届配置好后，启动代码

ros2 launch rm_vision_bringup vision_bringup.launch.py

ros2 launch rosbridge_server rosbridge_websocket_launch.xml



绿色打印表示识别到目标并且成功 KFC 预测
启动 Foxglove Studio,



具体串口数据看 rm_serial_driver

弹道解算

目标：保证弹丸能够精准击打目标

过程：

具体代码参照 algorithm_SolveTrajectory.cpp

首先确保 algorithm_SolveTrajectory.h 中的参数正确，包括弹道系数 k ，弹速 v ，偏置时间 $bias_times$, s_bias 和 z_bias 。

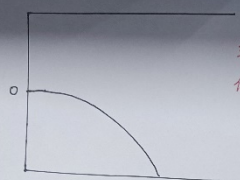
弄懂弹道解算的原理及其公式，参考小黑板的解算图，理解代码逻辑。

理解电控端弹道解算中的特别参数，如 Gyro->Yaw_angle 是陀螺仪 yaw 轴值等。

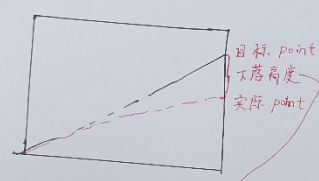
理解最终解算的误差值的原理及其逻辑，重点是 yaw。

调整静态补偿，保证误差值 New_Visual_Yaw 和 New_Visual_Pitch 在清零的情况下能够精确击打目标，包括远近高低，如果近距离打的中而远距离打不中的情况，要么就是相机标定出问题，要么就是陀螺仪飘了，要么就是弹道解算的逻辑或者是基本参数有问题需要仔细看，统一套参数不是适配所有自瞄调参，需要调整出适合的参数。

理想弹道模型
仅考虑了重力作用



横轴为 s , $s = \sqrt{x^2 + y^2}$, 纵轴为 z



目标 point
下落高度
实际 point

此弹道模型将利用迭代

设发射速度为 v_0 , 倾斜角

垂直 (z) $= v_0 t \cdot \sin \theta - \frac{1}{2} g t^2$

水平 (s) $= \sqrt{x^2 + y^2}$

空气阻力 model: $f = \frac{C_D \cdot \rho \cdot v^2}{2}$

C : 球体在空气中的摩擦系数

ρ : 空气密度, 温度由 0°C 到 25°C , 标准大气压取值为 1 .

S : 弹丸的接触面积

质量:

尺寸:

$v_s = v_0 \cdot \cos \theta$

$f_s = k_0 \cdot v_s^2$ ($k_0 = \frac{C_D \cdot S}{2}$)

$\frac{-f}{m} = a = \frac{dv_s}{dt}$

$\frac{-k_0 \cdot v_s^2}{m} = \frac{dv_s}{dt}$

$\frac{-k_0}{m} \cdot dt = \frac{dv_s}{v_s^2}$

$k_1 \cdot dt = \frac{-dv_s}{v_s^2}$ ($k_1 = \frac{k_0}{m}$)

积分得: $k_1 t + C = \frac{1}{v_s}$

由 $v_s(t=0) = v_{s0} = v_0 \cdot \cos \theta$ 得: $C = \frac{1}{v_{s0}}$

$\therefore v_s = \frac{v_{s0}}{k_1 \cdot v_{s0} \cdot t + 1}$

积分得: $s = \frac{1}{k_1} \cdot \ln(k_1 \cdot v_{s0} \cdot t + 1)$ (1)

迭代过程:

设 target (x, y, z), angle = pitch (计算仰角)

计算误差: $\text{delta } z = \text{target } z - \text{temp } z$

更新 temp Point = temp Point + delta Z

由 (1) 得: $t = \frac{e^{k_1 \cdot v_{s0} \cdot s} - 1}{k_1 \cdot v_{s0}}$

$z_{\text{actual}} = v_0 t \cdot \sin \theta - \frac{1}{2} g t^2$

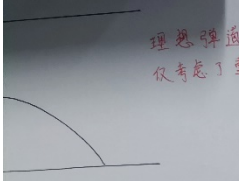
目标点 z_0 :

$\text{delta } z = z_0 - z_{\text{actual}}$ (直到 $\text{delta } z \approx 0$)

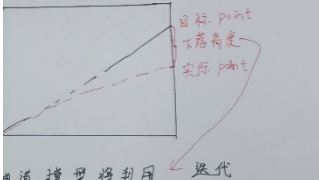
更新: $z_{\text{temp}} = z_{\text{temp}} + \text{delta } z \dots z_{\text{temp}}$ 收敛

2023.10.09 22:06

理想弹道模型
仅考虑了重力作用



横轴为 s , $s = \sqrt{x^2 + y^2}$, 纵轴为 z



目标 point
下落高度
实际 point

此弹道模型将利用迭代

设发射速度为 v_0 , 倾斜角为 θ

垂直 (z) $= v_0 t \cdot \sin \theta - \frac{1}{2} g t^2$

水平 (s) $= \sqrt{x^2 + y^2}$

空气阻力 model: $f = \frac{C_D \cdot \rho \cdot v^2}{2}$ (仅考虑水平空气阻力)

C : 球体在空气中的摩擦系数, 一般为 0.47

ρ : 空气密度, 温度由 0°C 到 25°C , 标准大气压取值为 1.293 kg/m^3 , 1.169 kg/m^3

S : 弹丸的接触面积

质量:

尺寸:

$v_s = v_0 \cdot \cos \theta$

$f_s = k_0 \cdot v_s^2$ ($k_0 = \frac{C_D \cdot S}{2}$)

$\frac{-f}{m} = a = \frac{dv_s}{dt}$

$\frac{-k_0 \cdot v_s^2}{m} = \frac{dv_s}{dt}$

$\frac{-k_0}{m} \cdot dt = \frac{dv_s}{v_s^2}$

$k_1 \cdot dt = \frac{-dv_s}{v_s^2}$ ($k_1 = \frac{k_0}{m}$)

积分得: $k_1 t + C = \frac{1}{v_s}$

由 $v_s(t=0) = v_{s0} = v_0 \cdot \cos \theta$ 得: $C = \frac{1}{v_{s0}}$

$\therefore v_s = \frac{v_{s0}}{k_1 \cdot v_{s0} \cdot t + 1}$

积分得: $s = \frac{1}{k_1} \cdot \ln(k_1 \cdot v_{s0} \cdot t + 1)$ (1)

迭代过程:

设 target (x, y, z), angle = pitch (计算仰角)

计算误差: $\text{delta } z = \text{target } z - \text{temp } z$

更新 temp Point = temp Point + delta Z

由 (1) 得: $t = \frac{e^{k_1 \cdot v_{s0} \cdot s} - 1}{k_1 \cdot v_{s0}}$

$z_{\text{actual}} = v_0 t \cdot \sin \theta - \frac{1}{2} g t^2$

目标点 z_0 :

$\text{delta } z = z_0 - z_{\text{actual}}$ (直到 $\text{delta } z \approx 0$)

更新: $z_{\text{temp}} = z_{\text{temp}} + \text{delta } z \dots z_{\text{temp}}$ 收敛

2023.10.09 22:06

陀螺仪检查

目标: 保证 pitch 轴和 yaw 轴的数据尽可能准确, 需要注意 yaw 轴或者 pitch 轴会不会相互影响, 以及有没有飘了导致大幅度变动。

过程:

代码查看：ros2 代码使用 topic 去查看，单独摆动 yaw 轴或者 pitch 轴，观察是否正常使用，不应该存在动 pitch 轴而 yaw 轴也在移动的情况或者在不动的情况下 yaw 轴持续变动的情况等，由于陀螺仪的特性（拉），需要长期进行检测。

原因：陀螺仪的数据准确会影响弹道解算的准确度。

自瞄使用

目标：电控端开启自瞄模式时可以正常使用（误差值清零的情况下精准击打，如果没有回到弹道解算那一步）

过程：

- 弹道解算（电控和视觉，最好都要懂逻辑，不能单方面的付出）

- 调 PID（电控部分）

- 陀螺仪数据的融合，保证 yaw 轴和 pitch 轴数据准确（电控部分）

- 自瞄模式的准确使用（电控和视觉）

简洁版的，后续需要进行改进，有问题就问（肯定会有问题的，问题是发现的，只会越来越多），发信息也行，打电话也行，有求必应。