



SJTU-VEX-AI进展汇报

汇报人：张远航



TABLE OF CONTENTS

- 01** 深度摄像头—视觉识别
- 02** 识别信息与主控通信
- 03** 灰度摄像头—全场定位
- 04** 未来发展规划

AI COMPETITION



深度摄像头——视觉识别





深度摄像头—视觉识别

1. RGB-D摄像头型号：Intel Realsense D435i
2. 程序环境：Ubuntu 18.04
3. 视觉算法：yoloV3/yoloV3 tiny
4. 模型训练：自动图像采集+图像增强处理+LabelImg人工标注+yoloV3/yoloV3 tiny
5. 返回数据：返回识别到的球的个数+每个球相对于摄像头的 (x, z) 坐标
6. 可视化：Opencv 4.0
7. 帧率：yoloV3 1.8fps; yoloV3 tiny 10fps
8. 识别效果：颜色太暗有概率无法识别，球体在图像内小于25%左右无法识别



视觉识别-yolo训练模型

识别对象：

- 1、RedBall 红色球
- 2、BlueBall 蓝色球
- 3、GreenBall 边界塔

训练准备：

安装darknet轻量深度学习框架，并且自己制作数据集并用python进行标定

训练工具：

使用yolo v3和yolo v3 tiny两个算法在交大学创的HPC云GPU上进行训练来得到两个权重文件

帧率优化：

使用TensorRT
提高识别帧率



视觉识别-效果展示



AI COMPETITION



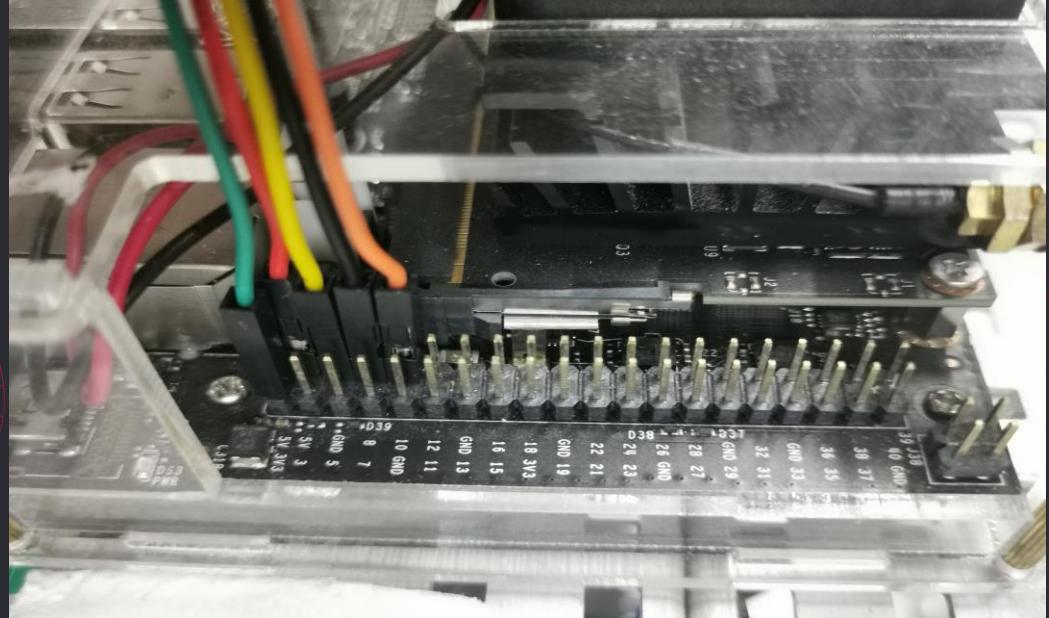
识别信息与主控通信





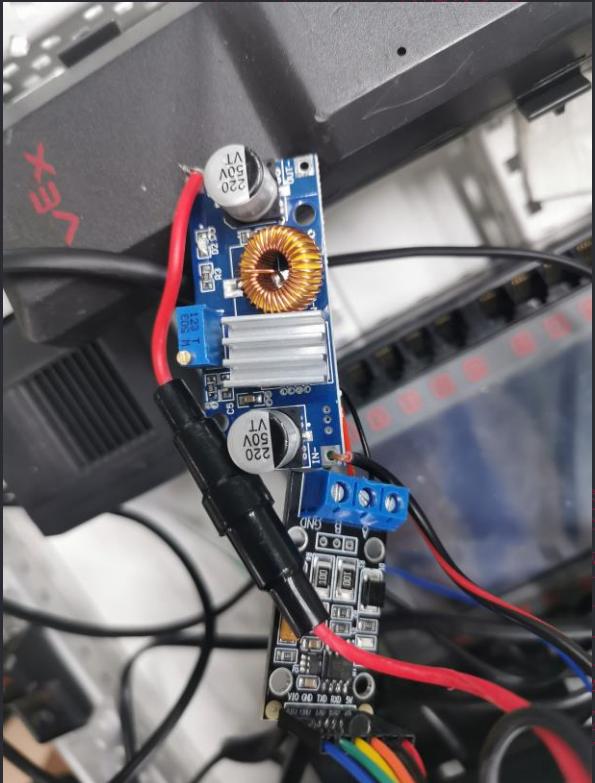
识别信息与*Brain*通信-串口通信

- 采用*Jetson Nano*开发板，其上有一系列引脚。官方预置了一些引脚的输出模块（参考：NVIDIA Jetson Nano J41Header Pinout - JetsonHacks），在这里我们使用 /dev/ttyTHS1 进行传输。如图所示，nano的5个引脚接出，8号和10号是 /dev/ttyTHS1 的信号传输引脚，5V和GND是给 nano 供电的引脚，3V3引脚是给转换模块供电，如右图：





识别信息与*Brain*通信-串口通信

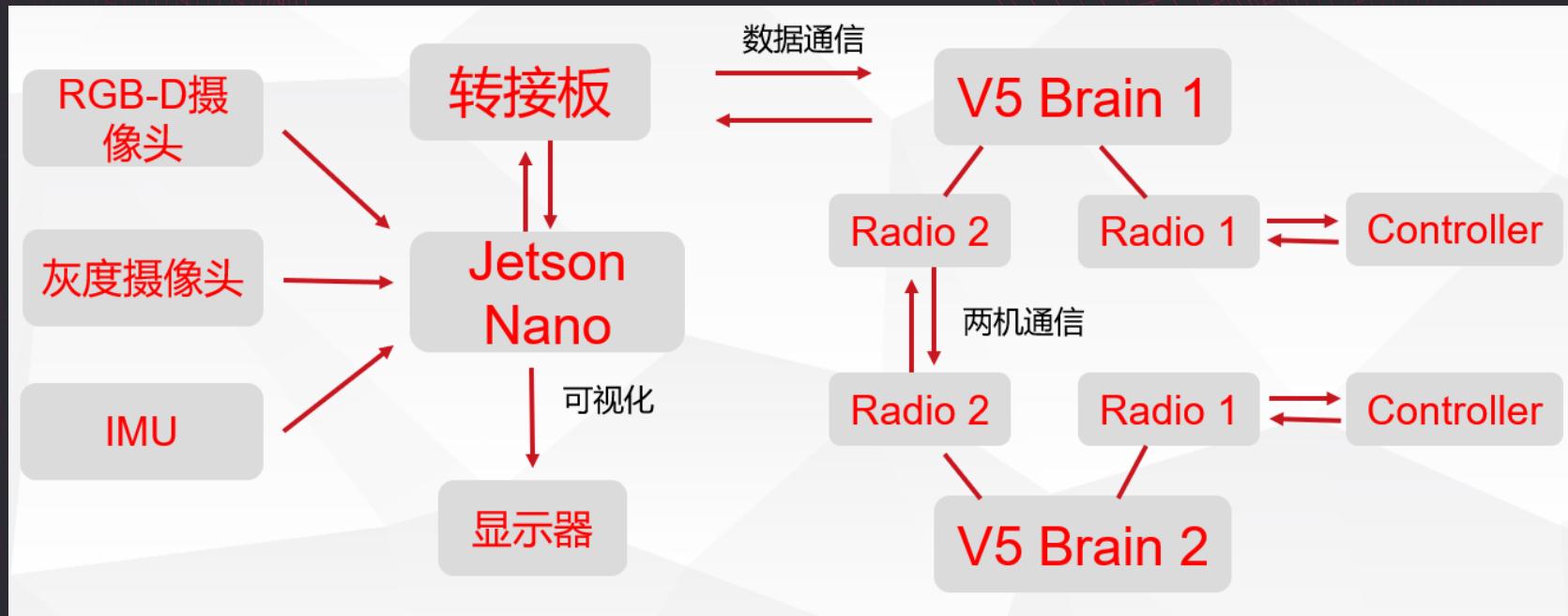


- 左方是变压模块，主要作用是将brain的马达口电压（12V）转化为nano合适的输入电压（5V），实现通过brain向nano供电。下方是装了一个**RS485 to TTL**的转换模块，**RS485**是vex brain端口的信号传输标准，**TTL**是**nano**输出的电平标准。在brain上，我们连接的是14号马达口（自定义的）。通过上述连接，我们可以实现brain对nano的供电，以及nano向brain传输信号。



识别信息与*Brain*通信-串口通信

- **brain上的解码：**设置缓冲区，每隔一段时间读取一次缓冲区的数据，读取数据后根据我们自定义的编码规则解码即可





识别信息与Brain通信—交互

1. 目的: Jeston Nano(后续会提到)获取了球的位置和颜色信息后，需要传输到vex brain，然后brain根据这些信息作出正确的反应，进而实现机器的自主运行

2. 技术难点:

- 1). nano识别到的球的位置信息是在“像素坐标系”的位置，如何将其转化为“相机坐标系”中的位置？
- 2). 如何将这些信息进行编码？
- 3). 如何将编码得到的数据流通过串口传输到brain？
- 4). 如何将物体识别和传输信号这两个代码模块相结合？
- 5). 如何在brain上进行解码，获得可用的数据？



识别信息与*Brain*通信-实现方法

ROS (Robot Operating System, 机器人操作系统) 提供一系列程序库和工具以帮助软件开发者创建机器人应用软件。它提供了硬件抽象、设备驱动、函数库、可视化工具、消息传递和软件包管理等诸多功能。

- 采用ROS:

1、通过ROS来获取相机的内参，并且根据得到的内参，进行准确的坐标变换。

2、ROS可以将相机、目标检测和通信三个节点同时运行。

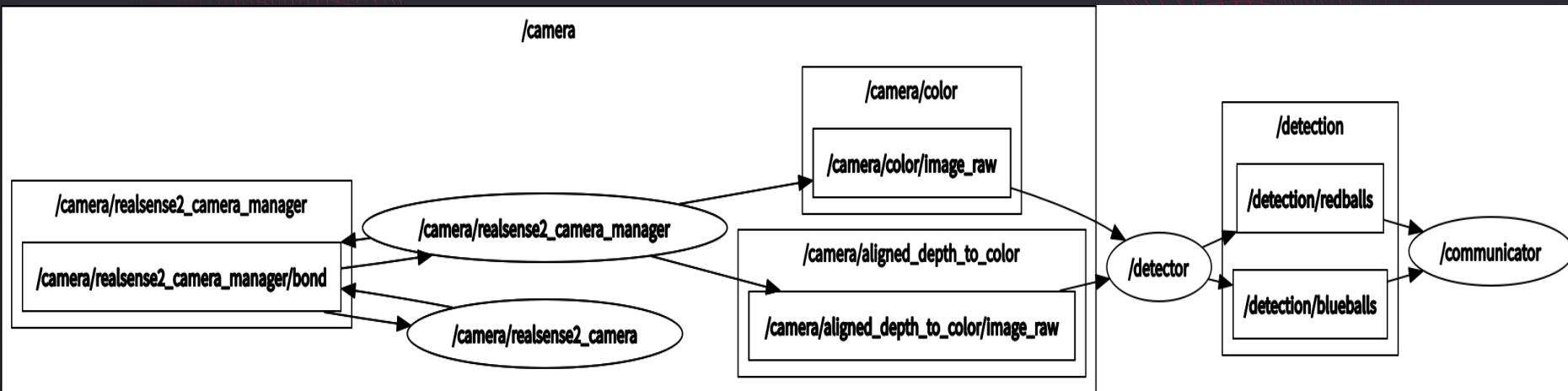
```
roslaunch realsense_detector detector.launch
```

3、ROS可以方便C++与Python混合编程



识别信息与*Brain*通信-实现方法

ROS以“话题发布—订阅”机制进行数据传输。相机模块 (camera) 得到的图像传给目标检测模块，目标检测模块 (detector) 识别图像后发布出两个话题redballs和blueballs到通信模块 (communicator) 。如下图所示。矩形代表话题，椭圆形代表模块。





识别信息与**Brain**通信-编码方式

- 每个 [] 代表一个字节

正常的数据包： (以下为方便阅读分行，实际上是以串起来为一个包)

```
[packet header 104]
  [ball]s message header 103]
    [red ball numbers]
      ([type header] [x value length] [x value byte by byte]s [type
header] [z value length] [z value byte by byte]s)s
    [blue ball numbers]
      ([type header] [x value length] [x value byte by byte]s [type
header] [z value length] [z value byte by byte]s)s
```

其中x是x坐标， z是深度（相机和球的距离）， 举例：

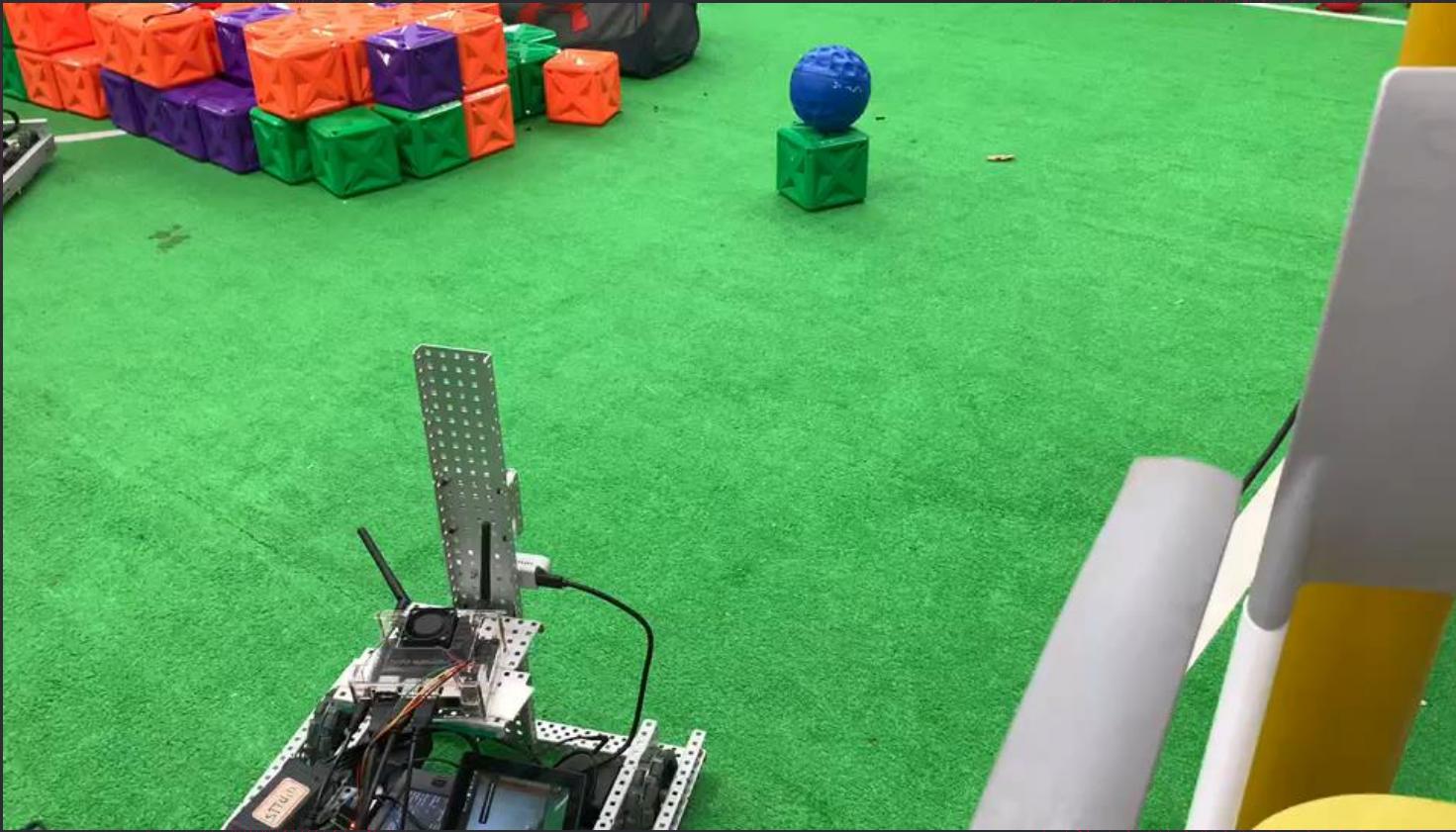
“hg\x01e\x050.110e\x050.598\x00”

- \x01 是red ball numbers， 表示检测到一个红球。 e是浮点数的type header， \x05 代表x value length是5， 后面的0.110则是x坐标值； 再往后， e还是浮点数的type header， \x05 代表z value length是5， 后面的0.598则是z坐标值。

最后的 \x00 是blue ball numbers， 表示没有检测到蓝色球。

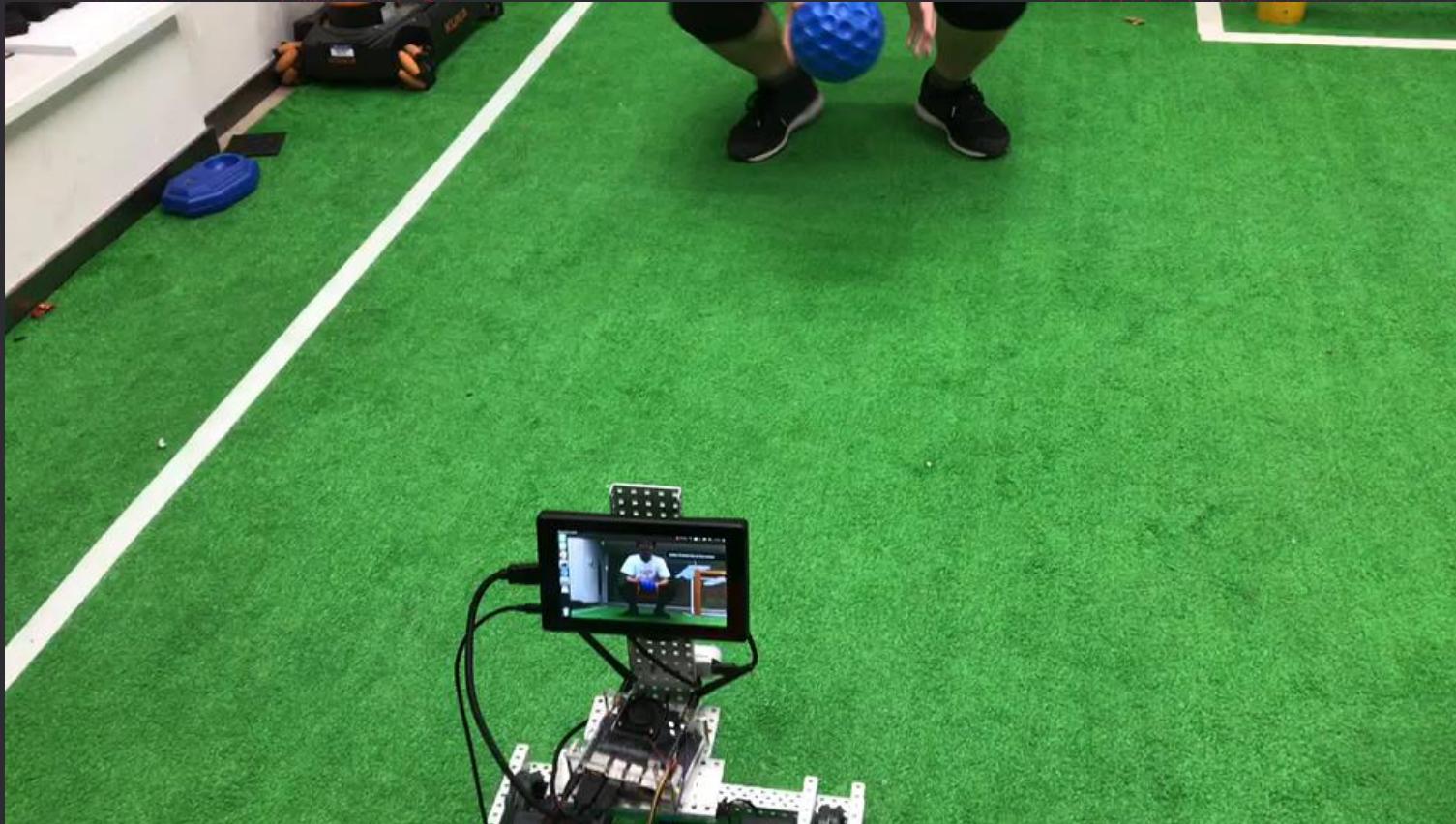


识别信息与*Brain*通信-效果展示





识别信息与*Brain*通信-效果展示



AI COMPETITION



灰度摄像头——全场定位





灰度摄像头—全场定位

1. 灰度摄像头型号：KS1A293
2. 所谓用库与可视化：Opencv
3. 使用算法：模糊匹配、单目视觉、卡尔曼滤波
4. 识别效果：距离边缘过远(大于场地的一半长度)就难以识别
5. 定位结果：较远时坐标误差在5cm—10cm



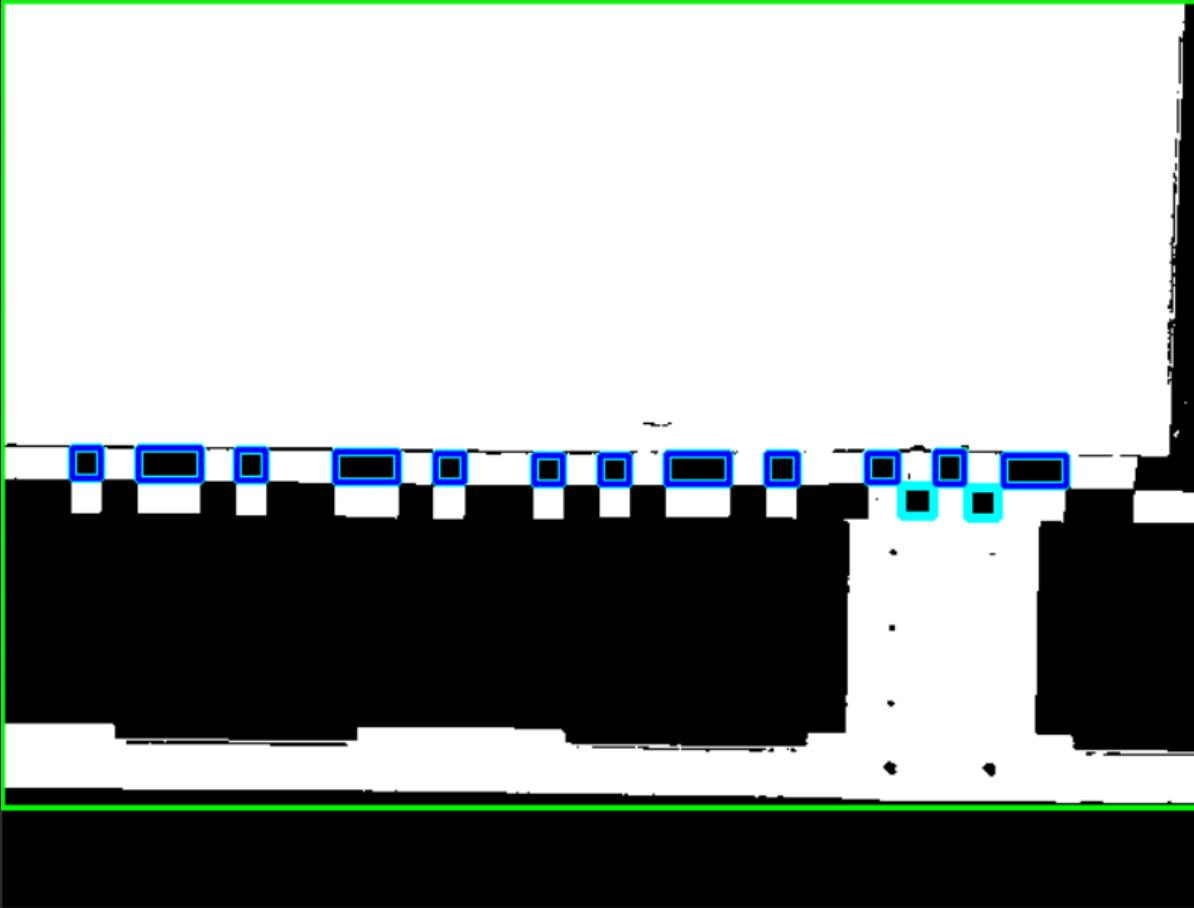
全场定位-识别算法

• 识别流程

- 1、对灰度图像进行透射变换与高斯滤波等预处理；
- 2、通过黑白条码中线上下灰度值在横向的变化检测提取中线；
- 3、通过黑白码大小以及位置特征对Opencv自动检测的contours进行滤波，得到目标黑白边框；
- 4、对部分缺失的黑白边框进行推测，来修复识别漏洞。



全场定位-识别效果展示



- 蓝色边框即为滤波后的边框，这里只需上层黑白码即可；



全场定位-识别匹配

- 匹配算法：

1、先将黑白框信息转化为**0-1**序列

2、随后运用模糊匹配算法，即：在所识别条码中出现部分小的识别错误仍然能有较大概率匹配成功：

```
# 模糊匹配
def fuzzyMatch(fieldList, getList):
    validlist, substartIndex = getValidList(getList)
    endIndex = substartIndex[-1] + len(validlist[-1]) - 1
    subLen = endIndex - substartIndex[0] + 1
    start = []
    endList = [1]
```



全场定位-匹配结果

- 最后得出所识别的条码序列在整个**fieldcode**中的起始位置等

如：

```
boxsize_zyh is: 15.416666666666666  
code is: [[0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, -1, -1, -1, -1, -1, 0, 0, 1, 1, 0, 1, 0]]  
angle now is: [-1.5707963267948966]  
startIndex is: 278.0  
completeGetList is: [0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 0, 1]  
vertices_TEST_0 is: [[14, 47, 240, 256], [65, 80, 241, 256], [117, 132, 241, 256], [152, 167, 241, 256], [186, 219, 240, 256], [238, 253, 241,  
index_start is: 278.0
```

- 1、boxsize_zyh：黑白格子边长，单位为像素格
- 2、code & completeGetList：前者为观测条码序列，后者为匹配到的条码序列(1为黑，0为白，-1为未识别到)
- 3、angle：陀螺仪所测的绝对转角
- 4、startIndex：匹配到的条码序列在整个fieldcode列表的开始下标



全场定位-坐标计算

- 单目视觉：

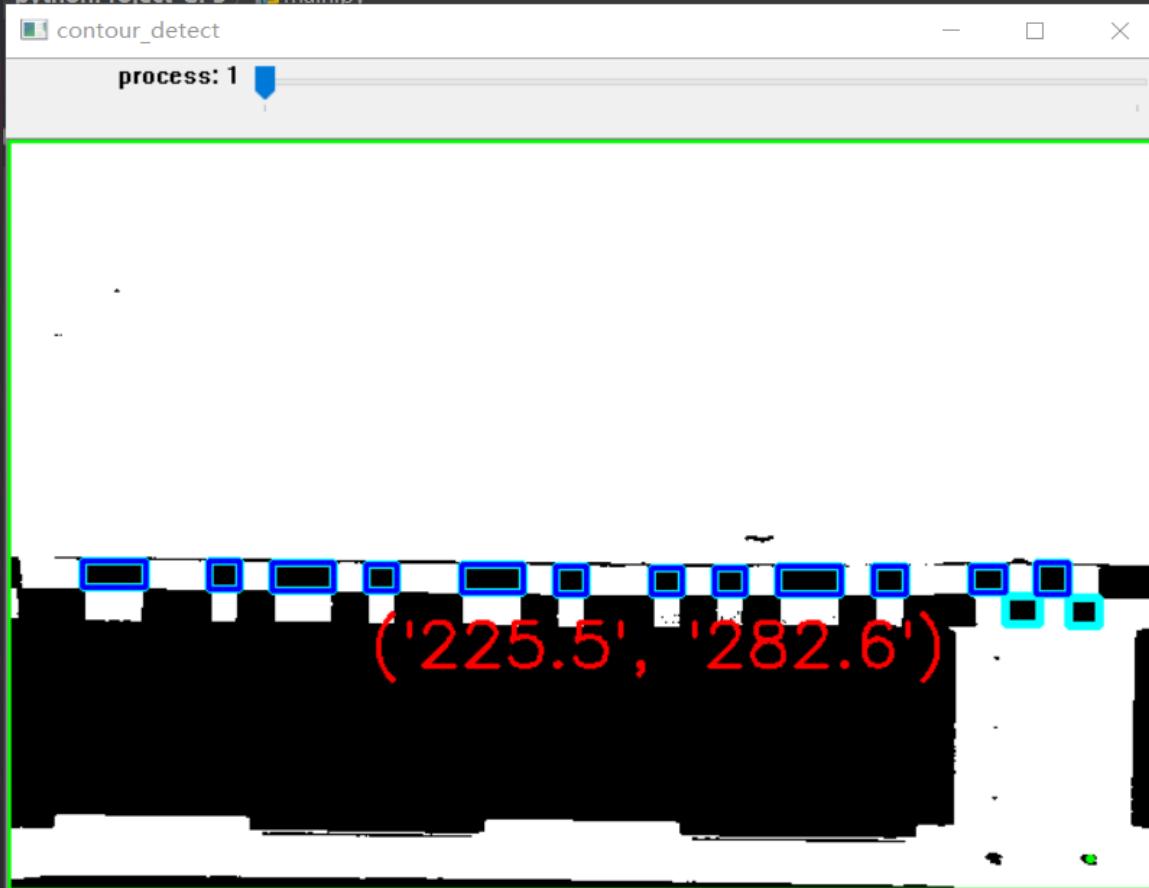
$$Z_c \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{1}{dx} & 0 & u_0 \\ 0 & \frac{1}{dy} & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} \vec{R} & T \\ 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} f_x & 0 & u_0 & 0 \\ 0 & f_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\text{相机内参}} \underbrace{\begin{bmatrix} \vec{R} & T \\ 0 & 1 \end{bmatrix}}_{\text{相机外参}} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

<http://blog.csdn.net/shenxwelling>

- (u, v) 为所匹配特定条码(首黑白格、尾黑白格)在像素坐标系下的坐标；
- (X_w, Y_w, Z_w) 为所匹配特定条码(首黑白格、尾黑白格)的世界坐标；
- Z_c 为所匹配特定条码(首黑白格、尾黑白格)相对于相机坐标系下的深度；
- 相机内参我们利用**matlab**进行张正友标定得到；
- R 为相机偏转角度， T 为相机方位， T 含有我们所需的相机坐标(即所求机器坐标，需反解)



全场定位-坐标计算效果



- 输出的是全局坐标(x, y)，单位是cm。



全场定位-坐标计算优化

- **卡尔曼滤波**: kalman滤波用在当测量值与模型预测值均不准确的情况下，用来计算预测真值的一种滤波方法。这在目标识别与追踪任务中经常用到。

```
# kalman init
last_mes = current_mes = np.array((2, 1), np.float32)
last_pre = current_pre = np.array((2, 1), np.float32)

kalman = cv2.KalmanFilter(4, 2)
kalman.measurementMatrix = np.array([[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]], np.float32)

def kalman_pos(x, y):
    global current_mes, mes, last_mes, current_pre, last_pre
    last_pre = current_pre
    last_mes = current_mes
    current_mes = np.array([np.float32(x), np.float32(y)])
```

- `last_mes & current_mes`: 上一帧与这一帧的测量(measure)值;
- `last_pre & current_pre`: 上一帧与这一帧的预测(predict)值。



全场定位-整体效果

The screenshot shows a development environment with a dark theme. On the left, there is a terminal window titled 'tk' displaying a black and white image with green bounding boxes around detected objects. Below the terminal is a file browser showing files like 'field code.txt' and 'gps filter.py'. On the right, there is a code editor with several tabs open: 'main', 'gps_run.py', 'gps_filter.py', '_main_.py', and 'record_testVideo.py'. The code in the 'main' tab includes functions for camera calibration and tracking, such as:

```
def calibrateCamera():
    # Calibration code using chessboard patterns

def track(frame):
    # Tracking code using contours and filters
    contours = findContours(frame)
    filterContours(contours)
    wrong = []
    for contour in contours:
        if filterContour(contour):
            wrong.append(contour)
    code_is = []
    for contour in wrong:
        if isCode(contour):
            code_is.append(contour)
    if len(code_is) > 0:
        track_back(code_is[0])
    else:
        track_back(wrong[0])
```

The status bar at the bottom shows build errors: 5 warnings and 15 errors.

AI COMPETITION

未来发展规划





未来发展规划

- **视觉识别：**目前对边塔的识别不够准确，后续需提高边塔识别精度然后加入，边塔的相关位置坐标信息；
- **全场定位：**需要提高识别的鲁棒性，即解决远距离识别问题；
- **通信策略：**承接视觉识别的进展加入边塔的通信信息到自定义的通信协议上；在完成基本功能的最后需要根据场地与机器信息写机器行驶策略。

AI COMPETITION

- 视觉识别：唐亚周 汤振宇 祝骥越 帅欣成 徐敬为 张一弛
- 全场定位：张远航 徐琳 曹家齐 徐敬为
- 通信：黄奕东 陈煜昂 郭春志
- 可视化：郑鸿晓

THANKS!

- 指导老师：楚朋志、武书昆



Does anyone have any questions?

mipzjy3000@sjtu.edu.cn

131 6237 2866

github.com/2020-SJTU-VEX