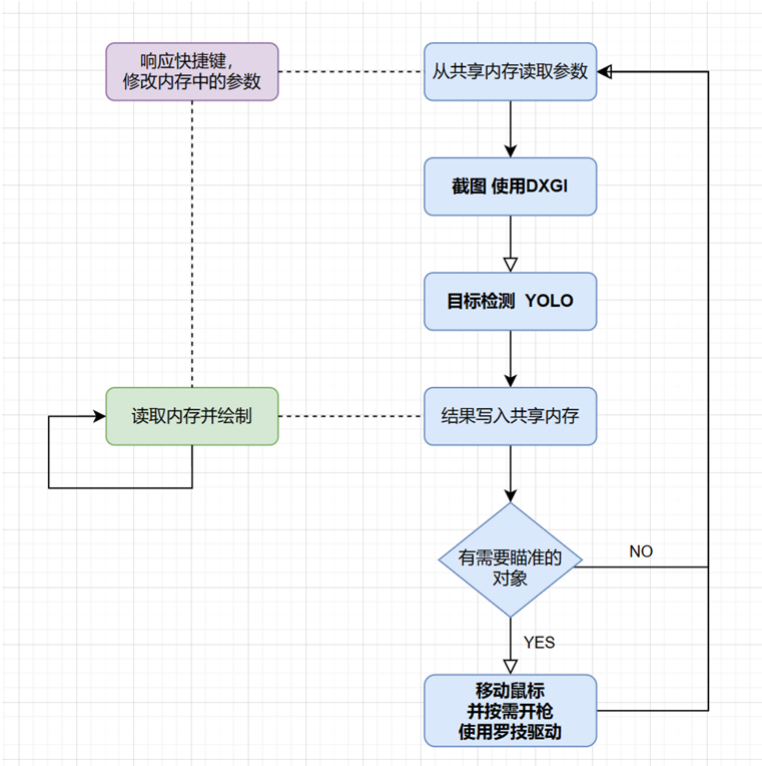
对于 demo_win_pytorch_normal\demo4.py

您可以查看从demo1到demo5的不同版本之间的变化

1.代码框架



2. 代码详解

基于multiprocessing共享内存，创建内存对象与多进程

```
if __name__ == '__main__':
    ct_heads_cnt = Value('i', 0, lock=False)
    ct_heads = Array('i', 20, lock=False)

    ct_bodys_cnt = Value('i', 0, lock=False)
    ct_bodys = Array('i', 20, lock=False)

    t_heads_cnt = Value('i', 0, lock=False)
    t_heads = Array('i', 20, lock=False)

    t_bodys_cnt = Value('i', 0, lock=False)
    t_bodys = Array('i', 20, lock=False)

    chickens_cnt= Value('i', 0, lock=False)
    chickens = Array('i', 20, lock=False)

    yes = Value('i', 0, lock=False) #是否开启功能
    k = Value('i', 0, lock=False) # 敌我识别,0表示均打,1表示打击t,2表示打击ct
    f = Value('i', 0, lock=False) #功能,0表示不自瞄,1表示锁身体,2表示锁头,3表示强制锁头(如果找不到头,根据身体计算头部位置)
    c = Value('i', 0, lock=False) #是否 自动开枪
    d = Value('i', 1, lock=False) # 是否 绘制图像
```

yes表示是否开启，k(kill)表示敌我识别， f(fun)表示功能， c(kfc)表示自动开枪，d(draw)表示绘制图像

其中，c的值为3表示强制锁头，即优先识别头，如果找不到头但找到身体（目标较远），则根据身体来计算头的位置

```
Get_data = Process(target=get_data, args=(
    yes, k, f, c,
    ct_heads_cnt, ct_heads,
    ct_bodys_cnt, ct_bodys,
    t_heads_cnt, t_heads,
    t_bodys_cnt, t_bodys,
    chickens_cnt, chickens
))

Get_key = Process(target=get_key, args=(yes, k, f, c, d))


Draw_screen = Process(target=draw_screen, args=(
    yes, d,
    ct_heads_cnt, ct_heads,
    ct_bodys_cnt, ct_bodys,
    t_heads_cnt, t_heads,
    t_bodys_cnt, t_bodys,
    chickens_cnt, chickens
))



Get_key.start()
Get_data.start()
Draw_screen.start()


Get_key.join()
Get_data.join()
Draw_screen.join()
```

Get_key 实现了快捷键

```
def get_key(yes,k,f,c,d):
    lst1=['均打击  ','仅打击t ','仅打击ct']
    lst2=['不自瞄  ','锁身体  ','锁头    ','强制锁头']
    lst3=['关闭','打开']
    lst4=['关闭','打开']
    def on_press(key):
        try:
            if key.char == 'o':
                if yes.value == 0:
                    print("//------------------------------------------------------------  ")
                    print('开启,参数为:')
                    print('敌我识别:'+lst1[k.value]+', 自瞄状态:'+lst2[f.value]+', 自动开枪:'+lst3[c.value]+', 绘制方框:'+lst4[d.value])
                    #print("  ------------------------------------------------------------//")

                    yes.value = 1
                elif yes.value == 1:
                    #print("//------------------------------------------------------------  ")
                    print('关闭,此轮检测信息为:')
                    yes.value = 0
            elif key.char == 'p':
                k.value+=1
                k.value%=3
                print('敌我识别:'+lst1[k.value]+', 自瞄状态:'+lst2[f.value]+', 自动开枪:'+lst3[c.value]+', 绘制方框:'+lst4[d.value])
            elif key.char == '[':
                f.value+=1
                f.value%=4
                print('敌我识别:'+lst1[k.value]+', 自瞄状态:'+lst2[f.value]+', 自动开枪:'+lst3[c.value]+', 绘制方框:'+lst4[d.value])
            elif key.char ==']':
                c.value+=1
                c.value%=2
                print('敌我识别:'+lst1[k.value]+', 自瞄状态:'+lst2[f.value]+', 自动开枪:'+lst3[c.value]+', 绘制方框:'+lst4[d.value])
            elif key.char =='\\':
                d.value+=1
                d.value%=2
                print('敌我识别:'+lst1[k.value]+', 自瞄状态:'+lst2[f.value]+', 自动开枪:'+lst3[c.value]+', 绘制方框:'+lst4[d.value])
```

on_press为处理键盘事件的函数，

键盘`o`(open)对应yes的值，

键盘`p`(person)对应k（敌我识别）的值，

键盘`[`代表f，键盘`]`代表c，键盘`\`代表d

当按下这些键的时候,修改共享内存中这值,并打印当前所有的状态.

Draw_screen 实现了方框绘制

在这个函数中,

我定义了一个draw_circle函数,这用于绘制共享内存中所有的信息一次

```
def draw_circle():
    canvas.delete("all")  # 删除先前的所有图案
    canvas.create_rectangle(0, 0, canvas.winfo_width(), canvas.winfo_height(), fill=TRANSCOLOUR, outline=TRANSCOLOUR)
    if yes.value==0 or d.value==0:
        time.sleep(1)
        return

    for i in range(0,chickens_cnt.value):
        canvas.create_rectangle(chickens[4*i],chickens[4*i+1],chickens[4*i+2],chickens[4*i+3],fill=TRANSCOLOUR,outline='black')

    for i in range(0,t_bodys_cnt.value):
        canvas.create_rectangle(t_bodys[4*i],t_bodys[4*i+1],t_bodys[4*i+2],t_bodys[4*i+3],fill=TRANSCOLOUR,outline='fuchsia')

    for i in range(0,ct_bodys_cnt.value):
        canvas.create_rectangle(ct_bodys[4*i],ct_bodys[4*i+1],ct_bodys[4*i+2],ct_bodys[4*i+3],fill=TRANSCOLOUR,outline='green')

    for i in range(0,t_heads_cnt.value):
        canvas.create_rectangle(t_heads[4*i],t_heads[4*i+1],t_heads[4*i+2],t_heads[4*i+3],fill=TRANSCOLOUR,outline='red')

    for i in range(0,ct_heads_cnt.value):
        canvas.create_rectangle(ct_heads[4*i],ct_heads[4*i+1],ct_heads[4*i+2],ct_heads[4*i+3],fill=TRANSCOLOUR,outline='blue')
```

同时，由于更改了窗口的属性，需要重新设置为不可点击

```
124         for i in range(0,ct_heads_cnt.value):
125             canvas.create_rectangle(ct_heads[4*i],ct_heads[4*i+1],ct_heads[4*i+2],ct_heads[4*i+3],fill=TRANSCOLOUR,outline='blue')
126         #设置窗口不可点击
127         hwnd = windll.user32.GetParent(tk.winfo_id())
128         exstyle = windll.user32.GetWindowLongW(hwnd, -20)
129         exstyle = exstyle | 0x80000 | 0x20
130         windll.user32.SetWindowLongW(hwnd, -20, exstyle)
```

update_circle函数用于定时调用draw_circle

```
def update_circle():
    draw_circle()
    tk.after(10, update_circle)
```

在主函数(draw_screen)中,我声明了一个tk窗口并设置其置顶透明, 并创建了一个canvas作为画板,随后调用update_circle函数.

```
ctypes.windll.shcore.SetProcessDpiAwareness(1)
ScaleFactor=ctypes.windll.shcore.GetScaleFactorForDevice(0)
tk = Tk()
tk.tk.call('tk', 'scaling', ScaleFactor/75)
TRANSCOLOUR = 'gray'
screen_width = tk.winfo_screenwidth()
screen_height = tk.winfo_screenheight()
print(screen_width,screen_height)
tk.attributes("-alpha", 1)
tk.geometry(bboxstr)
tk.overrideredirect(True)
tk.wm_attributes('-transparentcolor', TRANSCOLOUR)
tk.attributes("-topmost",True)
canvas = Canvas(tk)
canvas.pack(fill=BOTH, expand=Y)
update_circle()
tk.mainloop()
```

Get_data实现截图, 推理,移动鼠标

首先打开dll文件并声明罗技鼠标类

```
try:
    root = os.path.abspath(os.path.dirname(__file__))
    driver = ctypes.CDLL(f'{root}/logitech_driver.dll')
    ok = driver.device_open() == 1  # 该驱动每个进程可打开一个实例
    if not ok:
        print('Error, GHUB or LGS driver not found')
except FileNotFoundError:···

class Logitech:···
```

创建yolo模型,创建截图类的实例g,设置时间, 并打印初始设置信息

```
    model = YOLO(pt_path)
    g = DXGI.capture(*bbox)
    #m=mss.mss()
    cnt = 0
    t1 = time.time()
    #sum_time0=0
    sum_time1=0
    sum_time2=0
    sum_time3=0
    print('敌我识别:均打击  , 自瞄状态:不自瞄  , 自动开枪:关闭, 绘制方框:打开')
```

while循环来持续等待yes=1 ,并读取k、f、c的值并存储

```
    while(1):

        if yes.value == 0:
                time.sleep(1)
                continue
        elif cnt == 0:
            t1 = time.time()

        kk=k.value
        ff=f.value
        cc=c.value

```

进行50次识别:

首先截图并交由yolo推理，取回结果的tensor

```
        for t in range(50):
            cnt += 1
            #time.sleep(0.01)
            #begin_time=time.time()
            im = g.cap()
            # im=m.grab(bbox)
            # im=np.array(im)
            # im = cv2.cvtColor(im, cv2.COLOR_BGRA2BGR)

            # cv2.imshow('c', im)
            # cv2.waitKey(1)
            #im = cv2.resize(im, dsize=(640, 640))?

            results = model.predict(source=im, verbose=False, conf=0.65)
            ttmp=results[0].speed
            sum_time1+=ttmp['preprocess']
            sum_time2+=ttmp['inference']
            sum_time3+=ttmp['postprocess']
            cth=0 #ct的head的cnt为0
            ctb=0
            th=0
            tb=0
            chickensc=0
            b = results[0].cpu().boxes
            box = b.xyxy.numpy()
            cls = b.cls.numpy()
            conf =b.conf.numpy()
```

将张量移动到内存后写入共享内存中数据

```
            for i in range(len(cls)):
                #print(conf[i])
                if cls[i] == 0:
                    ct_heads[cth * 4] = box[i][0]
                    ct_heads[cth * 4+1] = box[i][1]
                    ct_heads[cth * 4+2] = box[i][2]
                    ct_heads[cth * 4+3] = box[i][3]
                    cth += 1
                elif cls[i]==1:
                    ct_bodys[ctb * 4] = box[i][0]
                    ct_bodys[ctb * 4+1] = box[i][1]
                    ct_bodys[ctb * 4+2] = box[i][2]
                    ct_bodys[ctb * 4+3] = box[i][3]
                    ctb += 1
                elif cls[i]==2:
                    t_heads[th * 4] = box[i][0]
                    t_heads[th * 4+1] = box[i][1]
                    t_heads[th * 4+2] = box[i][2]
                    t_heads[th * 4+3] = box[i][3]
                    th += 1
                elif cls[i]==3:
                    t_bodys[tb * 4] = box[i][0]
                    t_bodys[tb * 4+1] = box[i][1]
                    t_bodys[tb * 4+2] = box[i][2]
                    t_bodys[tb * 4+3] = box[i][3]
                    tb += 1
                else:
                    chickens[chickensc * 4] = box[i][0]
                    chickens[chickensc * 4+1] = box[i][1]
                    chickens[chickensc * 4+ 2] = box[i][2]
                    chickens[chickensc * 4+ 3] = box[i][3]
                    chickensc += 1
```

```
            ct_heads_cnt.value=cth
            ct_bodys_cnt.value=ctb
            t_heads_cnt.value=th
            t_bodys_cnt.value=tb
            chickens_cnt.value=chickensc
```

根据 k、f 的值， 遍历所有符合要求的数据，寻找离屏幕中心最近的目标，然后按需移动鼠标。

若鼠标离敌人已经很近，则开枪。

这里我在距离较远时快速移动，距离较近时慢速移动，是为了尽快加速对齐的速度，并避免震荡

每次分三次移动，是为了更真实模拟人类，并减少震荡

```
            if ff==0:
                #sum_time0+=time.time()-begin_time
                continue
            elif ff==1:
                if kk==0 or kk==2:
                    for i in range(ctb):
                        dx_new=(ct_bodys[4*i]+ct_bodys[4*i+2])/2+bbox[0]-bbox_mid[0]
                        dy_new=(ct_bodys[4*i+1]+ct_bodys[4*i+3])/2+bbox[1]-bbox_mid[1]
                        dis=dx_new*dx_new+dy_new*dy_new
                        if dis<d2:
                            dx=dx_new
                            dy=dy_new
                            d2=dis
                if kk==0 or kk==1:
                    for i in range(tb):
                        dx_new=(t_bodys[4*i]+t_bodys[4*i+2])/2+bbox[0]-bbox_mid[0]
                        dy_new=(t_bodys[4*i+1]+t_bodys[4*i+3])/2+bbox[1]-bbox_mid[1]
                        dis=dx_new*dx_new+dy_new*dy_new
                        if dis<d2:
                            dx=dx_new
                            dy=dy_new
                            d2=dis
            elif ff==2:…
            elif ff==3:…
            #print(d2)
            if d2==100000000:continue#未检测到目标
            if cc==1:
                if ff==1 and d2<100 or (ff==2 or ff==3)and d2<30:
                    Logitech.mouse.click(1)
            if d2<30000:
                # Logitech.mouse.move(int(dx), int(dy))
                Logitech.mouse.move(int(0.8*dx), int(0.8*dy))
                Logitech.mouse.move(int(0.3*dx), int(0.3*dy))
                Logitech.mouse.move(int(0.1*dx), int(0.1*dy))
            else:
                # Logitecsah.mouse.move(int(2*dx), int(2*dy))
                Logitech.mouse.move(int(1.6*dx), int(1.6*dy))
                Logitech.mouse.move(int(0.8*dx), int(0.8*dy))
                Logitech.mouse.move(int(0.3*dx), int(0.3*dy))
```

如果做完这50轮之后发现yes的值被置为0，说明我们“关闭”了模型，则输出结构并重置记录的信息。

随后进入while的下一个循环

```
        #sum_time0+=time.time()-begin_time
    #yes.value=0
    if yes.value == 0:
        t2 = time.time()

        print("计算了%d帧,检算帧率为%f" % (cnt, cnt / (t2 - t1)))
        print("每帧延时(即用时)%fms ,YOLO推理平均耗时%fms" % ((t2 - t1)*1000/cnt,(sum_time1+sum_time2+sum_time3)/cnt))
        #print("预处理耗时%fms,推理耗时%fms,后处理耗时%fms" % (sum_time1/cnt,sum_time2/cnt,sum_time3/cnt))
        print("  ------------------------------------------------------------//")
        sum_time1=0
        sum_time2=0
        sum_time3=0
        cnt=0
```

3. 选用的模型以及库的对比原因

yolo

关于yolo模型的选择，请参考ppt。

结论：对于我的设备，笔记本3060，

运行YOLOv8s时达到性能瓶颈（无论是识别准确度还是识别速度）

部署在TensorRT可以进一步提升性能约 25%

因此在最终的脚本里我选用了yolov8s并部署在Tensorrt上，

取得了最优的结果：约50帧（同时运行csgo2）

值得一提的是，yolo接受array格式的BGR图像，或者BCHW格式的RGB tensor

OpenCV	`cv2.imread('im.jpg')`	`np.ndarray`	HWC format with BGR channels `uint8 (0-255)`.
numpy	`np.zeros((640,1280,3))`	`np.ndarray`	HWC format with BGR channels `uint8 (0-255)`.
torch	`torch.zeros(16,3,320,640)`	`torch.Tensor`	BCHW format with RGB channels `float32 (0.0-1.0)`.

DXGI or mss?

运行test_cut_screen.py可得

dxgi速度为5ms，mss为12ms

```
(dxg) C:\Users\Limbo\Desktop\code\2024.3\AI\lab3>D:/A
e/2024.3/AI/lab3/test_files/test_cut_screen.py
0.0051088428497314455
0.012686645984649659
```

DXGI返回的是BGR的array数组，可以直接输入yolo

mss返回的是BGRA的Screenshot对象，需要转array数组后再转为BGR对象。

lgmouse or win32api ?

罗技鼠标驱动来移动鼠标消耗的时间<1ms，win32api的时间约为10m~20ms

但实测罗技鼠标dll移动后，似乎需要罗技驱动响应并反馈给windows。暂不清楚是否引入了额外的延时

tkinter

绘图很方便，但建议做一个异步处理来提高性能。