

山东大学（威海）
数据科学与人工智能实验班
重案六组

2022 年暑假大作业
小车无人驾驶
说明文档

组员：白锦帆，孙阳，李骁峰，杜海欣

感谢：引航计划

一、 自动驾驶流程综述

我们分成了以下几个部分来进行实现。

首先，我们的小组成员仔细研读了 PCB 的设计，并且进行了大量的讨论。结合引航计划所给出的代码和教程。我们搭建了该小车的网页控制端。使得我们能够通过浏览器来操控小车进行行走和拍照。随后，我们又根据银行计划的教程以及代码。搭建了一个没有网页页面的服务端。使得我们能够通过编写 Python 程序来控制小车。这就完成了小车的有人控制部分。为了实现小车无人驾驶，我们首先需要对我们的道路进行大量的采集。我们在小车的程序中直接写入环游程序，使得小车能够按部就班的按着道路进行，并且每隔一定时间进行拍摄，这样我们就得到了大量的样本图片。随后通过 EasyDL 对这些样本图片进行标注，并将其 mask 处理，并使用曙光计划的 GPU 进行模型训练，以此得到一个训练好的 Unet 语义分割模型。运用 Unet 语义分割模型将道路与非道路识别开来，再通过对小车摄像头传来回的照片进行处理，将其转化成一个鸟瞰图，形成能够被机器看懂的很多像素点。此时利用 Astar 算法来做出一个规划路径。设定好规划路径后，再根据纯追踪算法来确定小车具体应该做什么。

二、 如何让小车动起来

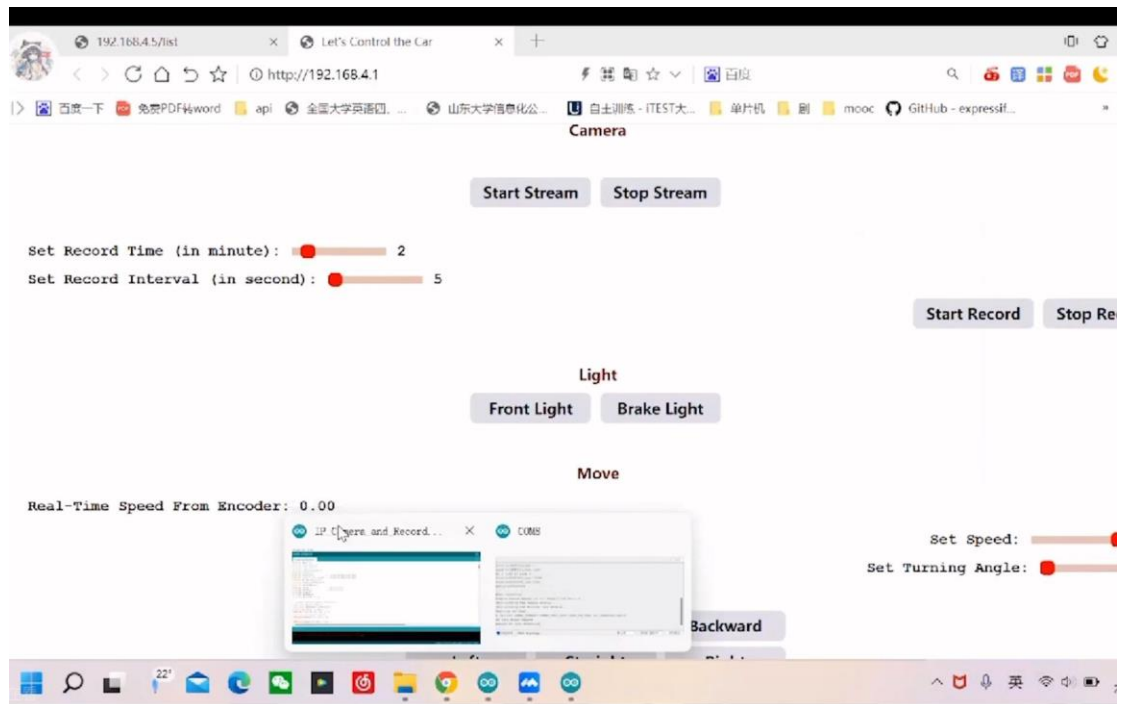
1、 通过 Web 控制

这里主要使用 arduino 制作一个网页，并且通过网页进行对小车的控制。这个控制分别是对于小车行进的控制和对于小车摄像头的控制。

首先我们需要下载一个 arduino 软件，设置开发板为“AI Thinker ESP32-cam”，一般情况下，下完软件后就可以在“工具”中选，如果没有的话可以查看一下软件的下载路径问题。然后在“文件”中找到“首选项”，将 https://dl.espressif.com/dl/package_esp32_index.json 添加到附加网址中，进行重启 arduino，然后在“工具”中的开发板管理器上，找到 ESP32 Arduino 下载，准备工作就可以了。

对于小车网页的代码，可以分为三部分：引脚的设置，网页的设置，和小车移动的函数。其中一定要注意 esp32 的 wifi 模块的设置，名称和密码都可以自己更改。而摄像头的代码方面，也一定要注意其中的网络设置要和小车网页部分的保持一致。

将两个代码分别上传到开发板模块上和摄像头上，在 arduino 上的串口监视器中查看 wifi，sd 卡等的连接状况。如若出现 ip 地址等信息，则可视成功。在打开网页时，注意将电脑也要连接在设置的网络上，这样才可以进行网页的操作。在看拍摄的照片时，如果相应设置的网页无法打开，在检查电脑的网络连接正确时，可以在网页后面加上/list 再打开。



2、 设定小车环游程序

在实现网页拍摄和存储照片之后，我们将要进行对地图道路的拍摄。这里有两种方法，一是在网页端控制小车的前进，后退，转弯，来实现对小车前进路线的规划，在网页设置拍摄的时间和频率，就可以获取到足够多的地图照片数量。第二种方法是通过调整小车舵机和电机的占空比，以及延时，来实现对小车前进路线的控制，同样在网页端设置好时间和频率。在list界面获取拍摄到的照片。如果list界面显示无储存，则需要利用读卡器进行照片读取。

```

void loop() {
  // put your main code here, to run repeatedly:
  while(1){
    ledcWrite(motorA_channel,128);
    delay(3000);
    ledcWrite(servo_channel,7);
    delay(1000);
    ledcWrite(motorA_channel,128);
    delay(3000);
    ledcWrite(servo_channel,20);
    delay(1000);
    ledcWrite(motorA_channel,128);
    delay(3000);
    ledcWrite(servo_channel,32);
    delay(1000);
    ledcWrite(motorA_channel,128);
    delay(3000);
    ledcWrite(servo_channel,20);
    delay(1000);
  }
}

```

3、 通过 Python 控制

在使用 Python 程序控制小车时，我们充分利用面向对象的编程思想以及银行计划的教程与代码。创建一个车辆控制类将有关方法写在其中，这样就可以方便地来进行控制。可以看到。这里有四个函数，他们都十分简单，只是像。192.168.4.1 发送相应的请求。例如，获得速度时就需要发送 read speed，这与我们的 arduino 代码中一样，获得角度时则相同，应当发送相应的请求。同时在获取这些数值之后，我们有些需要对它进行处理，例如实测速度向它发送 HTTP 请求时，返回的单位为厘米每秒，而我们需要的是毫米每秒的速度，所以我们就需要返回的是 speed 乘以十。同时，为了更好地控制车辆，我们还需要定义一个车辆状态类，这个状态类中有五个属性，他们代表车辆目前的状态。可以看到，这样我们就实现了使用 Python 程序来控制，或者说，我们就实现了使用键盘和鼠标来控制小车

```

class car_controller():

    def __init__(self):
        self.v = 0.0

    def motor_control(self, speed):
        # 这里的 speed 是电机的占空比, 不是实际速度
        requests.post('http://192.168.4.1/motor_control?speed={}'.format(speed))

    def servo_control(self, angle):
        # 这里的 angle 是舵机角度, 不是车的转角
        requests.post('http://192.168.4.1/servo_control?angle={}'.format(angle))

    def read_yaw(self):
        # 读取航向角, 单位度
        r = requests.get('http://192.168.4.1/read_angle')
        yaw = np.float16(r.text)

        return yaw

    def read_speed(self):
        # 读取实测速度 (单位为cm/s), 返回 mm/s 单位的速度
        r = requests.get('http://192.168.4.1/read_speed')
        speed = np.float16(r.text)

        return speed*10

```

```

'''
class VehicleState:# 定义一个类, 用于调用车辆状态信息

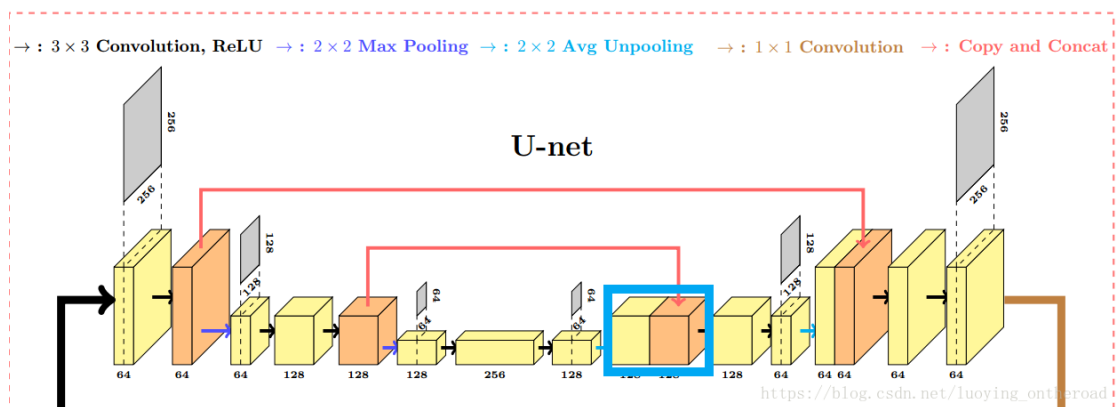
    def __init__(self, x=0.0, y=0.0, delta=0.0, yaw=0.0, v=0.0):
        self.x = x
        self.y = y
        self.delta = delta
        self.yaw = yaw
        self.v = v
'''

```

三、 如何让小车认识路

1、 Unet 语义分割

通过 Unet 语义分割模型, 识别小车摄像头的照片上的每一个像素点, 是道路还是草地, 还是背景, 以便进一步的操作。关于 Unet, 将在第四部分详细讲解。



2、转换为鸟瞰图

由于小车摄像头所采集到的是以小车摄像头为视角的图片，为了方便路径规划，我们需要将这些图片转化为鸟瞰图。

首先需要对图片进行去畸变处理，这是因为由于相机的特性，照片会有些许扭曲。

接着是对图片进行变换，这主要涉及以下两个矩阵，分别为内参和外参，通过这样的变化，可以将一张图片转化为鸟瞰图。

$$Z_c \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha & 0 & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

$$Z_c \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha & 0 & u_0 \\ 0 & \beta & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & u_0 & 0 \\ 0 & f_y & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

内参 外参

$$\begin{matrix} \text{内参} \\ \text{外参} \end{matrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} = K * M1 \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} = M \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

3、运用 Astar 算法寻路

Astar 算法，又称 A*算法，是一种静态路网中求解最短路径最有效的直接搜索方法。

公式表示为： $f(n)=g(n)+h(n)$,

其中， $f(n)$ 是从初始状态经由状态 n 到目标状态的最小代价估计，

$g(n)$ 是在状态空间中从初始状态到状态 n 的最小代价，

$h(n)$ 是从状态 n 到目标状态的路径的最小估计代价。

4、运用纯追踪算法

纯跟踪控制算法是一种典型的横向控制方法，最早于 1985 年提出，该方法

具有极强的稳定性。该方法的主要思想是在参考路径上自定义的距离 l_d 内匹配一个预瞄点，假设车辆后轮中心可以按照一定的转弯半径 R 行驶达到该预瞄点，然后根据预瞄距离转弯半径 R 车辆坐标下预瞄点的朝向角 2α 之间的几何关系来确定前轮转角。如图，根据三角几何关系，我们可以看到作为控制变量，前轮转角需要满足的关系。也就是角 δ ，它的正切值需要等于 R 分之 L 。联系以上两式，我们可以得到这个式子。由此，我们就知道了小车具体应该干什么。

算法精讲

- 在 $\triangle OAB$ 中, $AB \perp AO$, 则 $\angle AOC$:

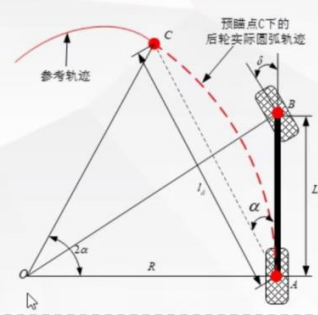
$$\angle AOC = \pi - 2\angle CAO = \pi - 2\left(\frac{\pi}{2} - \alpha\right) = 2\alpha \quad (1)$$
- 为了使车辆后轮跟踪圆弧虚线轨迹到达C点, 在 $\triangle OAC$ 中需要满足的正弦定理关系为:

$$\frac{l_d}{\sin 2\alpha} = \frac{R}{\sin\left(\frac{\pi}{2} - \alpha\right)} \quad (2)$$
- 化简上式, 得到:

$$R = \frac{l_d}{2\sin\alpha} \quad (3)$$
- 再看为了达到这种关系, 作为控制变量的前轮转角需要满足的关系。在阿克曼转向 (Ackermann turn) $\triangle OAB$ 中:

$$\tan\delta = \frac{L}{R} \quad (4)$$
- 因此, 联立上面两式, 得到:

$$\delta(t) = \arctan\left(\frac{2L\sin(\alpha(t))}{l_d}\right) \quad (5)$$



- 另外, 定义横向误差为车辆当前姿态和预瞄点在横向上的误差:

$$e_y = l_d \sin\alpha \quad (6)$$
- 联立(5)和 (6), 并考虑小角度假设, 有

$$e_y \approx \frac{l_d^2}{2L} \delta(t) \quad (7)$$
- 所以纯跟踪本质上是一个P控制器, 那么跟踪效果将由 l_d 决定, 通常定义 l_d 为关于速度的一次多项式:

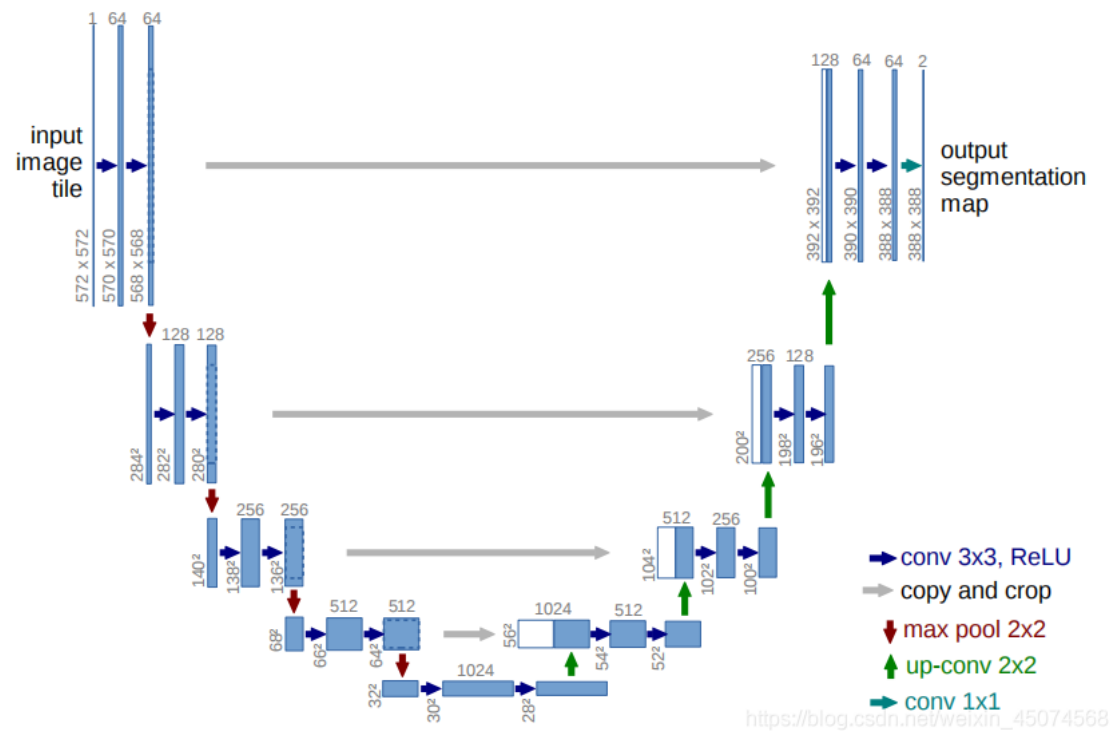
$$l_d = k_v v + l_{d0} \quad (8)$$

四、Unet 详解

1、什么是 Unet

接下来是对卷积神经网络 Unet 的介绍，Unet 是一个卷积神经网络，它被运用到语义分割等多种情形。所谓语义分割，就是将图片中的每一个像素识别出它所属的物体。例如，在一张有猫有狗的图片中，语义分割能够将属于猫的哪些像素标注出来，或者将属于狗的某些像素标注出来。在这里，我们通过语义分割来将道路识别出来，这样就可以让我们的小车看到道路。Unet 的主要贡献是在 u 型结构，该结构可以使它使用更少训练图片，同时且分割准确度也不会差。Unet 的网络结构如下。

它的左边是特征提取网络，它的右边是特征融合网络。



此为连续两次卷积

```
class DoubleConv(nn.Module):
    """(convolution => [BN] => ReLU) * 2"""

    def __init__(self, in_channels, out_channels, mid_channels=None):
        super().__init__()
        if not mid_channels:
            mid_channels = out_channels
        self.double_conv = nn.Sequential(
            nn.Conv2d(in_channels, mid_channels, kernel_size=3, padding=1, bias=False),
            nn.BatchNorm2d(mid_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(mid_channels, out_channels, kernel_size=3, padding=1, bias=False),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True)
        )

    def forward(self, x):
        return self.double_conv(x)
```

此为连续两次卷积再最大池化


```

class Down(nn.Module):
    """Downscaling with maxpool then double conv"""

    def __init__(self, in_channels, out_channels):
        super().__init__()
        self.maxpool_conv = nn.Sequential(
            nn.MaxPool2d(2),
            DoubleConv(in_channels, out_channels)
        )

    def forward(self, x):
        return self.maxpool_conv(x)

```

此为连续两次卷积再上采样

```

class Up(nn.Module):
    """Upscaling then double conv"""

    def __init__(self, in_channels, out_channels, bilinear=True):
        super().__init__()

        # if bilinear, use the normal convolutions to reduce the number of channels
        if bilinear:
            self.up = nn.Upsample(scale_factor=2, mode='bilinear', align_corners=True)
            self.conv = DoubleConv(in_channels, out_channels, in_channels // 2)
        else:
            self.up = nn.ConvTranspose2d(in_channels, in_channels // 2, kernel_size=2, stride=2)
            self.conv = DoubleConv(in_channels, out_channels)

    def forward(self, x1, x2):
        x1 = self.up(x1)
        # input is CHW
        diffY = x2.size()[2] - x1.size()[2]
        diffX = x2.size()[3] - x1.size()[3]

        x1 = F.pad(x1, [diffX // 2, diffX - diffX // 2,
                        diffY // 2, diffY - diffY // 2])
        # if you have padding issues, see
        # https://github.com/HaiyongJiang/U-Net-Pytorch-Unstructured-Buggy/commit/0e854509c2cea854e247a9c615f175f76fbb2e3a
        # https://github.com/xiaopeng-liao/Pytorch-UNet/commit/8ebac70e633bac59fc22bb5195e513d5832fb3bd
        x = torch.cat([x2, x1], dim=1)
        return self.conv(x)

```

此为最后的 1*1 卷积神经网络

```

class OutConv(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(OutConv, self).__init__()
        self.conv = nn.Conv2d(in_channels, out_channels, kernel_size=1)

    def forward(self, x):
        return self.conv(x)

```

此为特征提取网络

```

self.inc = DoubleConv(n_channels, 64 // n)
self.down1 = Down(64 // n, 128 // n)
self.down2 = Down(128 // n, 256 // n)
self.down3 = Down(256 // n, 512 // n)
factor = 2 if bilinear else 1
self.down4 = Down(512 // n, 1024 // factor // n)

```

此为特征融合网络

```

self.up1 = Up(1024 // n, 512 // factor // n, bilinear)
self.up2 = Up(512 // n, 256 // factor // n, bilinear)
self.up3 = Up(256 // n, 128 // factor // n, bilinear)
self.up4 = Up(128 // n, 64 // n, bilinear)
self.outc = OutConv(64 // n, n_classes)

```

此为 Unet 结构

```

def forward(self, x):
    x1 = self.inc(x)
    x2 = self.down1(x1)
    x3 = self.down2(x2)
    x4 = self.down3(x3)
    x5 = self.down4(x4)
    x = self.up1(x5, x4)
    x = self.up2(x, x3)
    x = self.up3(x, x2)
    x = self.up4(x, x1)
    logits = self.outc(x)
    return logits

```

2、 EasyDL 标注训练数据

接着对道路照片进行标注，这里用的是百度 easydl 平台训练道路分割深度学习模型。在网页端将照片另存为一个文件夹，来到 easydata 数据服务页面，选择创建数据集，并完成信息填写，然后导入拍摄到的照片。导入完成后进行对所有照片的标注，接着选择去训练，跳转到 easydl 平台，选择我的模型，创建一个新的模型，类型选择图像分类—道路分割，选择训练，数据集则选择之前标注完成的数据集，接着开始训练，结束后部署为云 api。然后我们可以回到 easydata，将完成标注的图片导出。



3、 mask 处理

为了方便神经网络处理图片，我们需要对图片的标注进行处理，将相对较为复杂的红色、绿色、黑色，处理成 0、1、2 的简单黑色。这只需要进行简单的数学运算。

```
grass_layer = rgb_mask[:, :, 1] / 128
background_layer = rgb_mask[:, :, 2] / 128
gray_mask = grass_layer + background_layer * 2
```

处理前的图片



处理后的图片

