目录

1. 实验背景简介

CCD（ChargeCoupledDevice）——电荷耦合器件，越来越广泛被应用于工业、军事、民用等行业，因其具有高解析度、低杂讯、动态范围广、良好的线性特性曲线等优点，在非接触测量及成像系统中应用极为广泛。面阵 CCD 是将像素元二维排列得到的。相较于线阵 CCD 而言，面阵 CCD 的前瞻性和全局性更好，获得的视觉信号更加稳定有效。由于在智能车竞赛中需要尽可能及时的控制转向、速度尽可能的快，使用面阵 CCD 能取得较好的效果。

本次实验基于 K66 开发板为核心板进行二次扩展开发的 Cyber-Dorm 实验板，进行编程实践，采集摄像头数据，实现对几种特征目标的识别，Cyber-Dorm 实验板照片如图 1-1 所示。

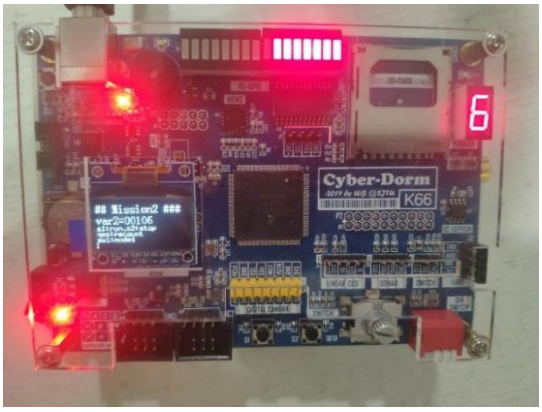


图 1-1 实验 Cyber-Dorm 开发板

2. 实验内容

本次实验需要完成的任务是使用DMA方式对MT9V034数字摄像头的信号进行采集并处理图像数据，以识别几种特征目标。

实验具体内容如下：使用DMA方式对MT9V034数字摄像头的信号进行采集，设计算法处理图像数据，识别直道、弯道、十字线、起始线、斑马线这几种特征目标，并识别“丢线”（场景中无有效赛道特征）情况且以光棒提示（蜂鸣器声音过吵，因而用光棒）。

实验任务
配置接口，对 MT9V034 数字摄像头的信号进行采集并显示
对原始图像进行处理，以给特征识别算法提供质量良好的图像
设计图像识别算法，识别特征目标

表格 2-1 实验任务表

3. 实验原理

3.1 面阵 CCD 工作原理

按一定的分辨率，以隔行扫描的方式采集图像上的点，当扫描到某点时，就通过图像传感芯片将该点处图像灰度转换成与灰度一一对应的电压值，然后将此电压值通过视频信号端输出。摄像头连续地扫描图像上的一行，则输出就是一段连续的电压信号，该电压信号的高低起伏反映了该行图像灰度变化。

当扫描完一行，视频信号端就输出一个低于最低视频信号电压的电平（如 0.3V ），并保持一段时间。这样相当于，紧接着每行图像信号之后会有一个电压“凹槽”，此“凹槽”叫做行同步脉冲，它是扫描换行的标志。然后，跳过一行后（摄像头是隔行扫描的），开始扫描新的一行，如此下去，直到扫描完该场的视频信号，接着会出现一段场消隐区。该区中有若干个复合消隐脉冲，其中有个远宽于（即持续时间远长于）其它的消隐脉冲，称为场同步脉冲，它是扫描换场的标志。场同步脉冲标志着新的一场的到来，不过，场消隐区恰好跨在上一场的 结尾和下一场的开始部分，得等场消隐区过去，上一场的视频信号才真正到来。

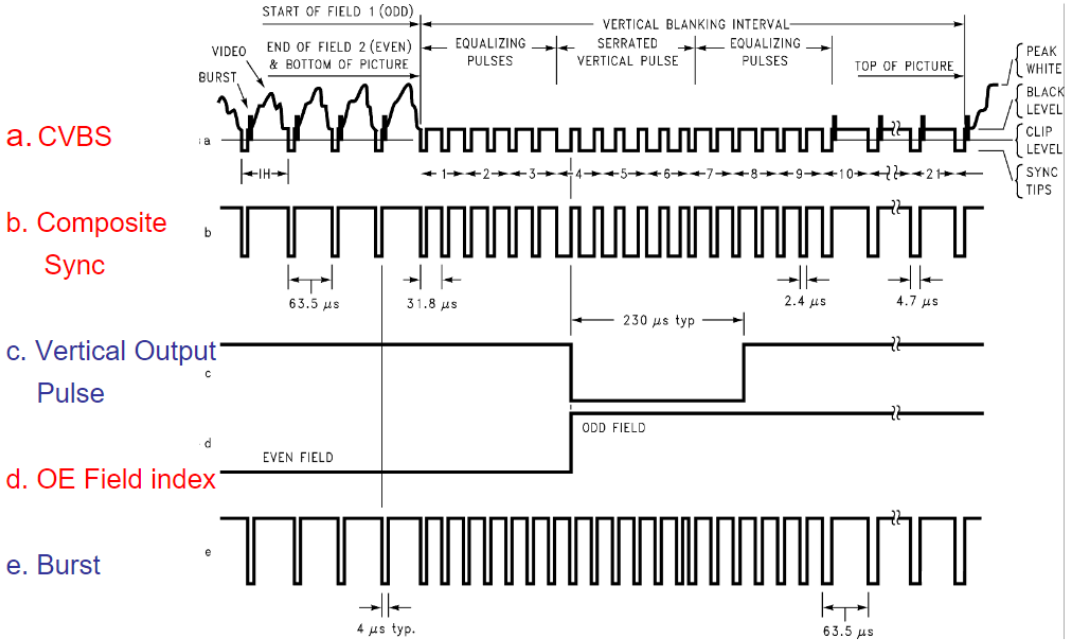


图 3-1 CVBS 同步信号时序关系图

3.2 DMA 工作原理

DMA：Direct Memory Access，直接存储器存取。使用 DMA 存取无需处理器干预，可

以在存储器与存储器或外设与存储器之间高速传输数据。

实现 DMA 传输时，是由 DMA 控制器直接掌管总线，因此，存在着一个总线控制权转移问题。即 DMA 传输前，CPU 要把总线控制权交给 DMA 控制器，而在结束 DMA 传输后，DMA 控制器应立即把总线控制权再交回给 CPU。

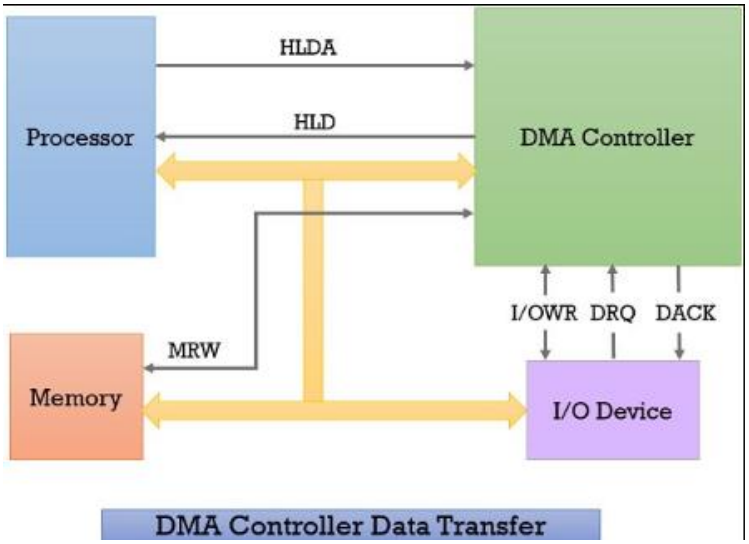


图 3-2 DMA 原理图

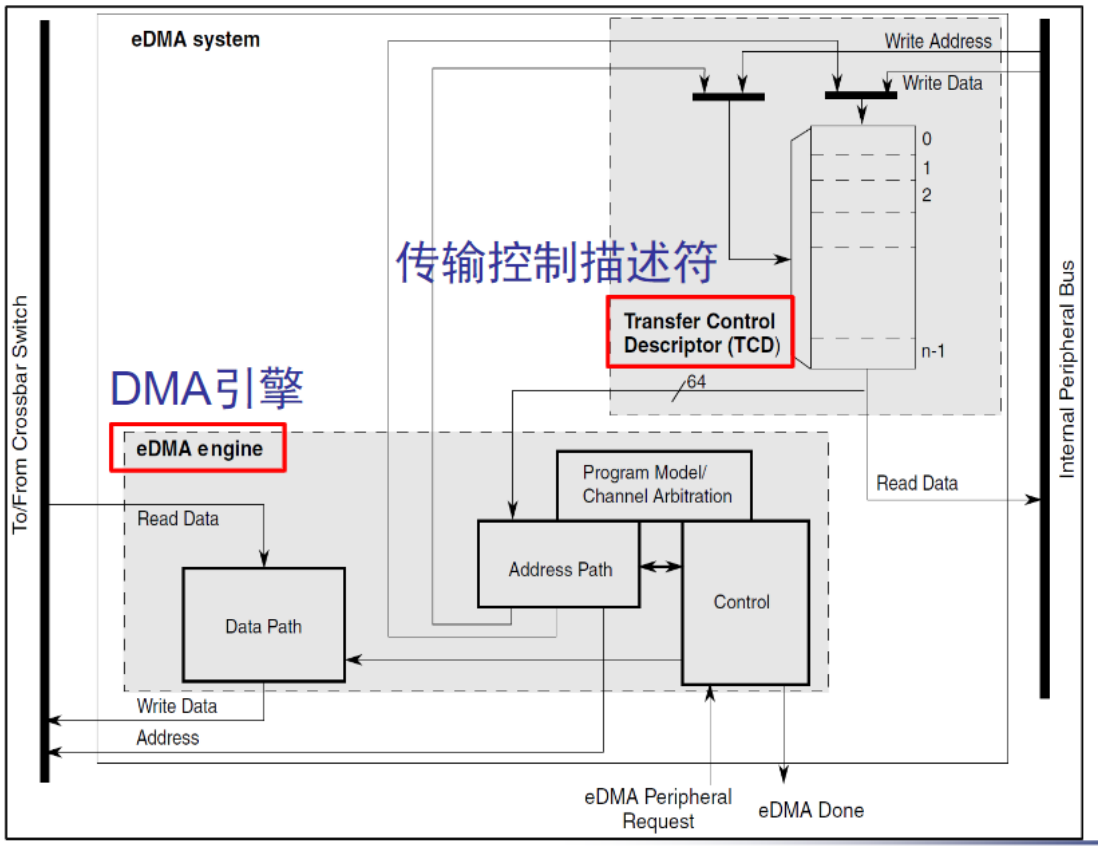


图 3-2 MK66 单片机 eDMA 结构框图

4. 实验方案

4.1 整体框架

本次实验的代码整体框架如下图 4-1 所示：

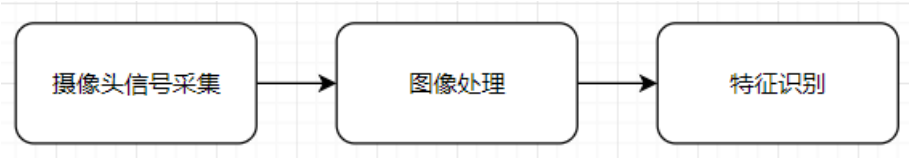


图 4-1 实验基本框图

我们小组利用基于 Ref6 示例代码文件进行开发拓展，完整实现了对 MT9V034 摄像头的信号采集和图像数据处理分析。

下文将实验分为了 DMA 采集数字摄像头信号、图像处理、寻线算法、特征识别四部分来进行陈述。

4.2 DMA 采集数字摄像头信号

使用 DMA 方式来完成采集数字摄像头信号。

DMA 全称为 Direct Memory Access（直接存储器存取）。使用 DMA 方式时，无需处理器干预，DMA 控制器管理数据传输和总线仲裁。DMA 方式可以实现在存储器与存储器或外设与存储器之间高速传输数据。DMA 基本配置参数包括：数据源地址、目标地址；触发源；数据字节宽度；主、副循环次数等。

使用 DMA 方式来完成采集数字摄像头信号的函数代码如下所示。按行采集摄像头每一像素的信号，每采集完一行（188 个像素）传输一次数据。每采集完一帧（120 行）会通知主循环得到了一帧完整的数据。

```
1.  /* PORTB_IRQn interrupt handler */
2.  void DCAM_IRQHANDLER(void) {
3.    /* Get pin flags */
4.    uint32_t pin_flags = GPIO_PortGetInterruptFlags(GPIOB);
5.    // HREF on PTB11
6.    if (pin_flags & BOARD_INITPINS_CAMHREF_GPIO_PIN_MASK)
7.    {
8.        GPIO_PortClearInterruptFlags(GPIOB, BOARD_INITPINS_CAMHREF_GPIO_PIN_MA
    SK);
9.        // Start DMA transfer on new line
10.           DMA_CAM_Repeat(CamLineCounter);
11.           CamLineCounter++;
12.           if (CamLineCounter >= 120)
13.           {
14.               CamLineCounter = 0;
```

```
15.           }
16.       return;
17.   }
18.   // VSYN on PTB16
19.   if (pin_flags & BOARD_INITPINS_CAMVSYN_GPIO_PIN_MASK)
20.   {
21.       GPIO_PortClearInterruptFlags(GPIOB, BOARD_INITPINS_CAMVSYN_GPIO_PIN_MA
    SK);
22.       CamDataReady = 1;
23.       CAM_DMA_Counter++;
24.       if ((CAM_DMA_Counter%50)==0)
25.           LED2_TOGGLE();
26.       return;
27.    }
28.
29.   // QESa on PTB20
30.   if (pin_flags & BOARD_INITPINS_QESa_GPIO_PIN_MASK)
31.   {
32.       GPIO_PortClearInterruptFlags(GPIOB, BOARD_INITPINS_QESa_GPIO_PIN_MASK)
    ;
33.       if (QESA())
34.       {
35.              if (!QESB()) QES_incflag = 1; //maybe increase
36.              if (QESB())  QES_decflag = 1; //maybe decrease
37.        }
38.        else
39.       {
40.            //Repeat judgment, elimination buffeting of keystroke of QES
41.            if ( QESB() && QES_incflag == 1)  {QESVar++;}
42.            if (!QESB() && QES_decflag == 1)  {QESVar--;}
43.            QESVar=MINMAX(0,QESVar,255);          //constrain(0,255)
44.            QES_decflag = 0;
45.            QES_incflag = 0;
46.        }
47.   }
48.
49.   /* Clear pin flags */
50.   GPIO_PortClearInterruptFlags(GPIOB, pin_flags);
51.
52.   /* Add for ARM errata 838869, affects Cortex-M4, Cortex-M4F
53.      Store immediate overlapping exception return operation might vector to
    incorrect interrupt. */
54.   #if defined __CORTEX_M && (__CORTEX_M == 4U)
```

```
55.     __DSB();
56.   #endif
57. }
58.
59. void DMA_CAM_Repeat(uint8_t LineNumber)
60. //void DMA_CAM_Repeat()
61. {
62.     DMA_DMA_BASEADDR->TCD[DMA_CH0_DMA_CHANNEL].DADDR          = (uint32_t) &I
    mage_Data[LineNumber][0];
63.     DMA_DMA_BASEADDR->TCD[DMA_CH0_DMA_CHANNEL].CITER_ELINKNO = DMA_CITER_ELI
    NKNO_CITER(188);
64.     DMA_DMA_BASEADDR->TCD[DMA_CH0_DMA_CHANNEL].BITER_ELINKNO = DMA_BITER_ELI
    NKNO_BITER(188);
65.     DMA_DMA_BASEADDR->SERQ = DMA_SERQ_SERQ(DMA_CH0_DMA_CHANNEL);
66. }
67.
68. /* DMA0_DMA16_IRQn interrupt handler */
69. void DMA_DMA_CH_INT_DONE_0_IRQHANDLER(void) {
70.
71.   /* Channel CH0 status */
72.   uint32_t CH0_status;
73.
74.   /* Reading all flags of status register */
75.   CH0_status = EDMA_GetChannelStatusFlags(DMA_DMA_BASEADDR, DMA_CH0_DMA_CHAN
    NEL);
76.   /* Clearing all flags of the channel status register */
77.   EDMA_ClearChannelStatusFlags(DMA_DMA_BASEADDR, DMA_CH0_DMA_CHANNEL, CH0_st
    atus);
78.   /* Add for ARM errata 838869, affects Cortex-M4, Cortex-M4F
79.      Store immediate overlapping exception return operation might vector to
    incorrect interrupt. */
80.   #if defined __CORTEX_M && (__CORTEX_M == 4U)
81.     __DSB();
82.   #endif
83. }
```

4.3 图像处理

受光线、噪声、畸变等因素影响，摄像头采集到的原始图像质量较差，无法直接用于特征识别。因此，我们对 MT9V034 反馈的图像进行了一系列的预处理与实验提取。

4.3.1 高斯滤波

高斯滤波是一种线性平滑滤波，适用于消除高斯噪声，广泛应用于图像处理的减噪过程。它对图像进行加权平均，每一个像素点的值,都由其本身和邻域内的其他像素值经过加权平均后得到，权值由二维高斯分布求得。

我们采用高斯滤波对反馈信息进行滤波预处理，并设计选用了3*3大小的高斯核，有效去除了部分图像噪声。

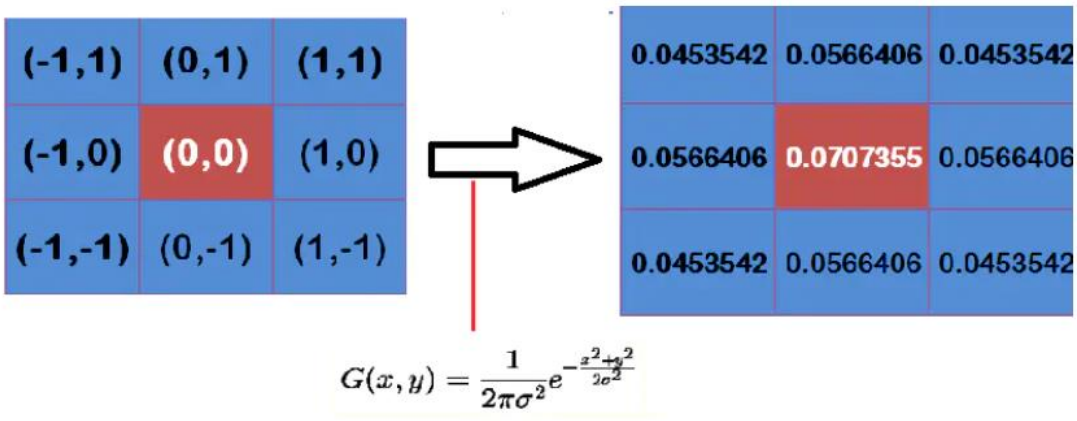


图4-2 高斯核示例及公式

实现了高斯滤波的函数代码如下。

```
1.  void fast_gauss_conv3(image_t input, image_t output){
2.      // convolution first line
3.      for(uint16_t r=1; r<MT9V034_IMAGEH2-1; r++){
4.          for(uint16_t c=1; c<MT9V034_IMAGEW2-1; c++){
5.              output[r][c] = input[r-1][c-1] * 4921 >> 16;
6.              output[r][c] += input[r-1][c] * 8113 >> 16;
7.              output[r][c] += input[r-1][c+1] * 4921 >> 16;
8.          }
9.      }
10.     // convolution second line
11.     for(uint16_t r=1; r<MT9V034_IMAGEH2-1; r++){
12.         for(uint16_t c=1; c<MT9V034_IMAGEW2-1; c++){
13.             output[r][c] += input[r][c-1] * 8113 >> 16;
14.             output[r][c] += input[r][c] * 13382 >> 16;
15.             output[r][c] += input[r][c+1] * 8113 >> 16;
```

```
        }
    }
    // convolution third line
    for(uint16_t r=1; r<MT9V034_IMAGEH2-1; r++){
        for(uint16_t c=1; c<MT9V034_IMAGEW2-1; c++){
            output[r][c] += input[r+1][c-1] * 4921 >> 16;
            output[r][c] += input[r+1][c] * 8113 >> 16;
            output[r][c] += input[r+1][c+1] * 4921 >> 16;
        }
    }
}
```

4.3.2 动态阈值二值化

由于待识别场景为颜色分明的黑白标志，因此可直接采用阈值二值化的做法来将灰度图转化为二值图，便于后续提取分析。

二值化的基本原理为：灰度高于阈值的像素点在二值图中表示为1，否则为0。常规二值化方法由于静态阈值的设定，无法适用于各类环境，这里我们选用了随着光照等外界环境的变动作自适应变化的动态阈值算法进行二值化运算。

4.3.2.1 大津法

大津法是一种自适应的阈值确定的方法。它是按图像的灰度特性，将图像分成背景和目标 2 部分。背景和目标之间的类间方差越大，说明构成图像的 2 部分的差别越大，当部分目标错分为背景或部分背景错分为目标都会导致 2 部分差别变小。因此，使类间方差最大的分割意味着错分概率最小。大津法可以对类间方差为单峰的图像产生较好的分割效果，但对噪音和目标大小十分敏感。

在线性 CCD 传感器的使用中，对于 CCD 采集到的一维图像，将车道两侧的黑线作为前景，白色 KT 板作为背景，分割阈值记为 T。前景像素点的比例为 a_0，平均值为 b_0，背景像素点的比例为 a_1，平均值为 b_1。图像的总平均值为 b，类间方差为 g。可以得到

$$g = a_0 * a_1 \ (b_0 - b_1)^2$$

从最小值到最大值遍历 g，当 g 取得最大值时，可以认为此时前景和背景的差异最大，此时的灰度值 b 是最佳阈值。通过大津法，能够求出一个阈值，用这个阈值进行图像二值化分割后，前景与背景图像的类间方差最大，能够有效地分离标志图案和背景。

实现了大津法的函数代码如下。

```
uint8_t GetOTSU(uint8_t image[MT9V034_IMAGEH2][MT9V034_IMAGEW2])
{
    uint16_t i, j;
    uint32_t Amount = 0;
    uint32_t PixelBack = 0;
    uint32_t PixelIntegralBack = 0;
    uint32_t PixelIntegral = 0;
    uint32_t PixelIntegralFore = 0;
```

```
9.      uint32_t PixelFore = 0;
10.     float OmegaBack, OmegaFore, MicroBack, MicroFore, SigmaB, Sigma; // 类间方差;
11.     uint16_t MinValue, MaxValue;
12.     uint8_t Threshold = 0;
13.     uint8_t HistoGram[256];
14.
15.     for (j=0; j<256; j++)  HistoGram[j] = 0; //初始化灰度直方图
16.
17.     for (j=0; j<MT9V034_IMAGEH2; j++)
18.     {
19.         for (i = 0; i < MT9V034_IMAGEW2; i++)
20.         {
21.             HistoGram[image[j][i]]++; //统计灰度级中每个像素在整幅图像中的个数
22.         }
23.     }
24.
25.     for (MinValue = 0; MinValue < 256 && HistoGram[MinValue] == 0; MinValue++) ;        //获取最小灰度的值
26.     for (MaxValue = 255; MaxValue > MinValue && HistoGram[MinValue] == 0; MaxValue--) ; //获取最大灰度的值
27.
28.     if (MaxValue == MinValue)     return MaxValue;          // 图像中只有一个颜色
29.     if (MinValue + 1 == MaxValue)  return MinValue;         // 图像中只有二个颜色
30.
31.     for (j = MinValue; j <= MaxValue; j++)    Amount += HistoGram[j];        //  像素总数
32.
33.     PixelIntegral = 0;
34.     for (j = MinValue; j <= MaxValue; j++)
35.     {
36.         PixelIntegral += HistoGram[j] * j;//灰度值总数
37.     }
38.     SigmaB = -1;
39.     for (j = MinValue; j < MaxValue; j++)
40.     {
41.         PixelBack = PixelBack + HistoGram[j];    //前景像素点数
42.         PixelFore = Amount - PixelBack;          //背景像素点数
43.         OmegaBack = (float)PixelBack / Amount;//前景像素百分比
44.         OmegaFore = (float)PixelFore / Amount;//背景像素百分比
45.         PixelIntegralBack += HistoGram[j] * j;  //前景灰度值
```

```
46.         PixelIntegralFore = PixelIntegral - PixelIntegralBack;//背景灰度值
47.         MicroBack = (float)PixelIntegralBack / PixelBack;   //前景灰度百分比
48.         MicroFore = (float)PixelIntegralFore / PixelFore;   //背景灰度百分比
49.         Sigma = OmegaBack * OmegaFore * (MicroBack - MicroFore) * (MicroBack - MicroFore);//计算类间方差
50.         if (Sigma > SigmaB)                    //遍历最大的类间方差 g //找出最大类间方差以及对应的阈值
51.         {
52.             SigmaB = Sigma;
53.             Threshold = j;
54.         }
55.     }
56.     return Threshold;                        //返回最佳阈值;
57.
58. }
```

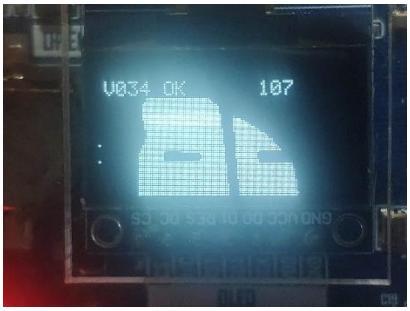
4.3.2.2 平均值阈值法

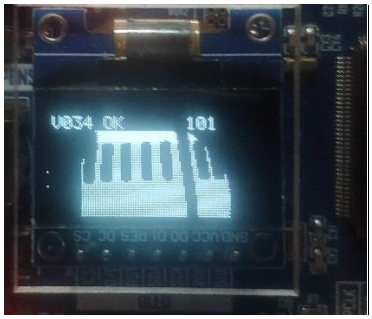
平均值阈值法是根据图像灰度的均值确定二值化阈值的简易方法。
实现了平均值阈值法的代码如下。

```
1.  for (i=0; i<MT9V034_IMAGEH2; i++)
2.  {
3.       for (j=0; j<MT9V034_IMAGEW2; j++)  tv += pic[i][j];   //累加
4.  }
5.  Threshold = tv/MT9V034_IMAGEH2/MT9V034_IMAGEW2;
6.  Threshold = Threshold*8/10+10;
```

4.3.3 融合差分图像与二值化图像

经过动态阈值二值化后的图像如下图所示，可以看到二值化后的图像有着较好的区分标志的能力，但在边缘处由于摄像头畸变及其它原因，有着大块的黑斑。

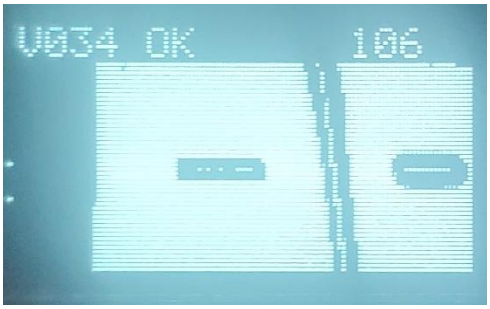




这里，我们采用了融合差分边缘算子的方法，将二值图与差分图像进行融合，有效去除了黑色斑点，也去除了一些误识别的噪声。

差分图像就是目标场景在连续时间点图像相减所构成的图像，假设x_k和x_{k+1}分别为目标场景在时间点k和k+1时的图像，则可以定义第k幅差分图像为：$\Delta x_k = x_{k+1} - x_k$。由于差分图像能够表现出图像中图形的边缘，融合后的二值图能更准确地表示场景的特征，不容易出现图形残缺的问题。

处理之后的图像如下所示，与之前直接二值化得到的图形相比，融合差分算子后的算法已基本足以满足我们的需求。



实现了差分图像与二值化图像融合的代码如下。

```
1.  for (i=1; i<MT9V034_IMAGEH2-1; i++)
2.      {
3.          for(j=1; j<MT9V034_IMAGEW2-1; j++)
4.          {
5.              if(binary_pic[i][j] == 0)
6.              {
7.                  if(origin_pic[i][j]-origin_pic[i-
    1][j]<diff && abs(origin_pic[i][j]-origin_pic[i+1][j])<diff
8.                   && origin_pic[i][j]-origin_pic[i][j-
    1]<diff && abs(origin_pic[i][j]-origin_pic[i][j+1])<diff     )
9.                  {
10.                     binary_pic[i][j] = 1;
11.                 }
12.             }
13.         }
14.     }
```

4.3.4 开运算去除噪点

开运算是对二值图先腐蚀再膨胀的形态学处理方式，能够消除噪点。

腐蚀是使用模板（卷积核）与图像进行卷积运算，得出模板覆盖区域的像素点最小值并将这个最小值代替参考点的像素值。

膨胀同腐蚀操作相似，但它是扫描图像中的每一个像素点，用模板元素与二值图像元素做“与”运算，如果结果都为零，则参考像素点为零，否则为一。

实现了开运算的函数代码如下。

```
void erode3(image_t input, image_t output){
    uint8_t tmp0=0,flag=0;
    for(uint16_t r=1; r<MT9V034_IMAGEH2-1; r++){
        for(uint16_t c=1; c<MT9V034_IMAGEW2-1; c++){
            output[r][c] = 0;
            flag=0;
            for(uint16_t i=0; i<9; i++)
            {
                tmp0 = input[r+dr_3[i]][c+dc_3[i]];
                if(tmp0==1) flag++;
            }
            if(flag>6)
            {
                output[r][c] = 1;
            }
        }
    }
}

void dilate3(image_t input, image_t output){
    uint8_t tmp0,flag=0;
    for(uint16_t r=1; r<MT9V034_IMAGEH2-1; r++){
        for(uint16_t c=1; c<MT9V034_IMAGEW2-1; c++){
            output[r][c] = 1;
            flag=0;
            for(uint16_t i=0; i<9; i++){
                tmp0 = input[r+dr_3[i]][c+dc_3[i]];
                if(tmp0==0) flag++;
            }
            if(flag>5)
            {
                output[r][c] = 0;
            }
        }
    }
}
```

4.3.5 canny 边缘提取算法

在本次实验中，曾经尝试直接对高斯处理后的图像直接进行canny边缘提取算法，希望能直接得到图形的轮廓。但发现效果较差，可能是由于图像噪点过多的原因，因此放弃了这一方案。

Canny边缘提取算法主要是以下步骤：首先使用高斯滤波器平滑图像，然后利用一阶

偏导的有限差分计算梯度的幅值和方向，在之后对梯度幅值进行非极大值抑制，最后利用双阈值算法检测和连接边缘。

Canny边缘提取算法重点是计算梯度强度和方向和非极大值抑制两步。

计算梯度强度和方向可以使用Sobel算子进行计算。Sx 和Sy 即为Sobel算子。

$$G_x = S_x * A = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} * \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} = sum\left(\begin{bmatrix} -a & 0 & c \\ -2d & 0 & 2f \\ -g & 0 & i \end{bmatrix} \right)$$

$$G_y = S_y * A = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} = sum\left(\begin{bmatrix} a & 2b & c \\ 0 & 0 & 0 \\ -g & -2h & -i \end{bmatrix} \right)$$

图4-3 x y方向梯度计算示例

非极大值抑制算法为：将当前像素的梯度强度与沿正负梯度方向上的两个像素进行比较。如果当前像素的梯度强度与另外两个像素相比最大，则该像素点保留为边缘点，否则该像素点将被抑制。

Canny边缘提取算法的实现函数代码如下。

```
1.  void Canny(image_t A, image_t B, uint8_t low_thres, uint8_t high_thres){
2.      static image_t CannyAm, Sector;
3.      int Xg = 0; //X 方向梯度
4.      int Yg = 0; //Y 方向梯度
5.      uint8_t loss = 2;
6.      uint8_t temp;
7.      for (int i = 0; i < MT9V034_IMAGEH2; i++)
8.      {
9.          for (int j = 0; j < MT9V034_IMAGEW2; j++)
10.         {
11. //          Xg=(A[i][j]+A[i+1][j]-A[i+1][j+1]-A[i][j+1])>>1;
12. //          Yg=(-A[i][j]+A[i+1][j]-A[i][j+1]+A[i+1][j+1])>>1;
13.             // sobel kernel in the horizontal direction
14.             Xg  = ((int)A [i-1][j+1]
15.                 - (int)A [i-1][j]
16.                 + ((int)A[i+0][j+1]<<1)
17.                 - ((int)A[i+0][j]<<1)
18.                 + (int)A [i+1][j+1]
19.                 - (int)A [i+1][j]) >> 1;
20.
21.             // sobel kernel in the vertical direction
22.             Yg  = ((int)A [i][j-1]
```

```
                + ((int)A[i][j+0]<<1)
                + (int)A [i][j+1]
                - (int)A [i+1][j-1]
                - ((int)A[i+1][j+0]<<1)
                - (int)A [i+1][j+1]) >> 1;
            CannyAm[i][j] = sqrt(Xg*Xg+Yg*Yg); //求幅值，快速开平方算法
            Sector[i][j] = Atan2(Yg, Xg);  //求梯度方向分区
        }
    }
    //非极大抑制
    for(int y=1; y<MT9V034_IMAGEH2-1; y++)
    {
        for(int x=1; x<MT9V034_IMAGEW2-1; x++)
        {
            switch(Sector[y][x]){

            case 0: //水平方向
                if( CannyAm[y][x] < (CannyAm[y][x+1] - loss) || CannyAm[y][x] < (CannyAm[y][x-1] - loss) )
                {
                    CannyAm[y][x] = 0;
                }
                break; // end of 0
            case 1: //右上、左下
                if( CannyAm[y][x] < (CannyAm[y-1][x+1] - loss) || CannyAm[y][x] < (CannyAm[y+1][x-1] - loss) )
                {
                    CannyAm[y][x] = 0;
                }
                break; //end of 1
            case 2: //竖直方向
                if( CannyAm[y][x] < (CannyAm[y-1][x] - loss) || CannyAm[y][x] < (CannyAm[y+1][x] - loss) )
                {
                    CannyAm[y][x] = 0;
                }
                break; //end of 2
            case 3: //左上、右下
                if( CannyAm[y][x] < (CannyAm[y-1][x-1] - loss) || CannyAm[y][x] < (CannyAm[y+1][x+1] - loss) )
                {
                    CannyAm[y][x] = 0;
                }
```

```
                break; //end of 3
            }
        } //end of for(x)
    } // end of for(y)


    for(int y=1; y<MT9V034_IMAGEH2-1; y++)
    {
        for(int x=1; x<MT9V034_IMAGEW2-1; x++)
        {
            if( CannyAm[y][x] < low_thres ) //低于低阈值
            {
                B[y][x] = 255;
            }
            else if( CannyAm[y][x] > high_thres ) //高于高阈值
            {
                B[y][x] = 0;
            }
            else
            {
                temp = A[y+1][x-1];
                if(temp < A[y+1][x]) temp = A[y+1][x];
                if(temp < A[y+1][x+1]) temp = A[y+1][x+1];
                if(temp < A[y][x-1]) temp = A[y][x-1];
                if(temp < A[y][x]) temp =A[y][x];
                if(temp < A[y][x+1]) temp = A[y][x+1];
                if(temp < A[y-1][x-1]) temp = A[y-1][x-1];
                if(temp < A[y-1][x]) temp = A[y-1][x];
                if(temp < A[y-1][x+1]) temp = A[y-1][x+1];
                if(temp > high_thres)
                {
                    B[y][x] = 0;
                    CannyAm[y][x] = 0;
                } //end of if
                else
                {
                    B[y][x] = 255;
                    CannyAm[y][x] = 255;
                 }
             }
         }
     }
}
```

4.4 寻线算法

寻线算法的作用是把图像变成由线组成的骨架，便于后续识别特征。

4.3.5 左右记分寻线法

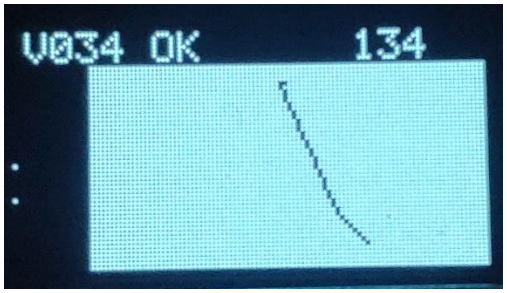
在本次实验中，左右寻线法的基本原理是首先确定寻线的初始行数与位置，之后以上一行的位置为基础进行左右寻线，并可根据置信度给予位置一定的评分形成左右记分寻线法。由于在上一次线性CCD的处理中，我们采用了这一算法，因而本次选用了另一种常用的算法DFS寻线算法。

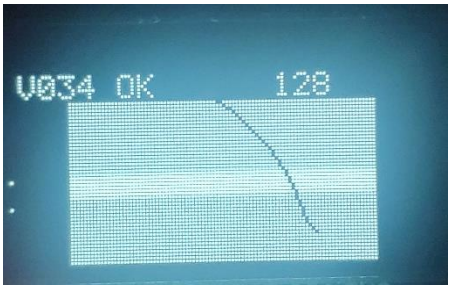
4.3.5 DFS 算法

DFS 算法是编程中常用的一种实验算法，其用于寻线的优先是，存在反馈回溯，因而可以稳定寻找到最长的线段，也就是我们需要跟踪的线段。

实验发现，常规 DFS 算法在单线寻找上较为可靠且基本足以得到正确的骨架线条。

DFS 算法寻线的结果图如下所示。





DFS（深度优先搜索）寻线实现的函数代码如下。

```
const int dx[] = {-1,  0,  1, -1, 1,};
const int dy[] = {-1, -1, -1,  0, 0,};
const uint32_t d_size = sizeof(dx) / sizeof(dx[0]);
#define ROAD_LINE_LENGTH_MIN    (30)
uint16_t length[MT9V034_IMAGEH2][MT9V034_IMAGEW2];

void reset_dfs(){
    memset(length, 0, sizeof(length));
}

uint32_t dfs_find(image_t edge, uint8_t x, uint8_t y, uint8_t len){
    uint8_t _x, _y, remain=0, temp;
    len++;
    length[y][x] = len;
    for(uint16_t i=0; i<d_size; i++){
        _x = x + dx[i];
        _y = y + dy[i];
        if(0<=_x && _x<MT9V034_IMAGEW2 && 0<=_y && _y<MT9V034_IMAGEH2 && edge[_y][_x]==0 && length[_y][_x]==0){
```

```
            temp = dfs_find(edge, _x, _y, len);
            if(temp > remain) remain = temp;
        }
    }
    length[y][x] = len + remain;
    return remain + 1;
}

extern uint32_t line[200][2];
void dfs_save(uint8_t x, uint8_t y, uint8_t size){
    uint8_t len = length[y][x], _x, _y;
    length[y][x] = 0;
    size--;
    if(size==1) return;
    line[size][0] = x;
    line[size][1] = y;
   // printf("%d,%d\n",x,y);
    //printf("%d,%d\n",line[size][0],line[size][1]);
    for(uint32_t i=0; i<d_size; i++){
        _x = x + dx[i];
        _y = y + dy[i];
        if(0<_x && _x<MT9V034_IMAGEW2 && 0<_y && _y<MT9V034_IMAGEH2 && length[_y][_x]==len && size>0){
            dfs_save( _x, _y, size);
        }
    }
}


void find_road_lines(image_t edge, uint32_t line[][2], int32_t *len){
    uint32_t _len=0;
    reset_dfs();
    for(uint16_t y=MT9V034_IMAGEH2 - 10; y>=10; y--){
        for(uint16_t x=MT9V034_IMAGEW2-10; x>=10; x--){
            _len = dfs_find(edge, x, y, 0);
            if(_len > ROAD_LINE_LENGTH_MIN){
                if(_len < *len) *len = _len;
                dfs_save(x, y, *len);
                goto found;
            }
        }
    }
    *len = 0;
```

```
61.   found:
62.       return;
63. }
64.
65. void draw_road_line(image_t road,uint32_t len){
66.     memset(road, 1, sizeof(image_t));
67.     for(uint32_t i=0; i<len; i++)
68.     {
69.         if(5<line[i][1] && line[i][1]<MT9V034_IMAGEH2-
    5 && 5<line[i][0] && line[i][0]<MT9V034_IMAGEW2-5)
70.         {
71.             road[line[i][1]][line[i][0]] = 0;
72.         }
73.     }
74. }
```

4.5 特征识别

根据给定的几种场景特征的特点，我们设计了基于分布特性和差分特性的识别算法，并以此为原理进行了调参修正，最终形成了较为稳定的特征识别算法。

4.5.1 基于分布特性的识别

斑马线，十字线，起始线这三种图形比较复杂的场景特征，有较为独特的分布特性，所以用分布特性即可把它们辨识出来。

4.5.2 基于差分特性的识别

两种不同的直线以及曲线属于比较简单的图形，分布特性相对不明显，因此要结合每行寻到的路线的差分特性才能辨识出它们。

对于直线 1 而言，各行的差分是一个相对稳定的较小常值，而对于直线 2 而言，各行的差分为稳定的较大常数，而对于弯道线段，差分可得到一个变化的数值，且这一数值在最后几行会有较大的改变

对所有类型的场景特征都尝试识别之后，如果没有发现任何一种特征，则视为“丢线”，蜂鸣器短鸣提示。

4.5.3 特征参数设定

应老师要求，我们在识别特征后，定量设置速度及方向参数作为特征参数。

	速度参数	方向参数
路况 1：直线 1	匀速行驶	直行

路况 2：直线 2	根据直线斜率进行减速	根据特征转向
路况 3：弯道	根据弯道曲率进行减速	根据特征转向
路况 4：十字路	减速冲过	直行
路况 5：起始线	加速冲出	直行
路况 6：斑马线	停车	直行

对于这些过程，我们给予一定的标识参量并展示在 oled 屏上，并使用蓝色 led 灯闪烁的频率表示当前输出速度，用光棒显示输出舵角，当未识别特征时，便输出 0 状态，光棒全亮（由于蜂鸣器声音较大，为避免对周围人造成干扰，我们选用光棒指示）。

整个特征识别过程的代码实现如下。

```
1.  void road_judge(image_t binary)
2.  {
3.      uint16_t last_col=0,last_col5,mean_count;
4.      mean_count=0;
5.      image.meanx=0;
6.      uint32_t tmp_line[MT9V034_IMAGEH2];
7.      for(uint16_t h=3; h<MT9V034_IMAGEH2-3; h++)
8.      {
9.        image.row[h]=0;
10.       image.points=0;
11.       image.diff[h]=100;
12.       image.diff5[h]=100;
13.
14.       for(uint16_t w=3; w<MT9V034_IMAGEW2-3; w++)
15.       {
16.           if(binary[h][w]==0)
17.           {
18.               image.points++;
19.               image.row[h]++;
20.               tmp_line[h]= w ;
21.               if(h>15 && h<45 && w>5 && w<90)
22.               {
23.                  image.meanx=image.meanx+w;
24.                  mean_count++;
25.               }
26.               if(h>9)
27.               {
28.                  image.diff[h]=w-last_col;
29.                  last_col=w;
30.                  if(h%5==0)
31.                   {
32.                      image.diff5[h]=w-last_col5;
```

```
                    last_col5=w;
                }
            }
        }
    }
    image.meanx=image.meanx/mean_count;
    uint16_t most_num[100];
    uint16_t nor_num=0,max_tmp=0,diff_count=0;
    float mean_num=0,sum_num=0;
    for(uint16_t h=10; h<MT9V034_IMAGEH2-10; h++)
    {
        if(image.diff[h]<100)
        {
          most_num[image.diff[h]]++;
          most_num[image.diff[h]+1]++;
          most_num[image.diff[h]-1]++;
          diff_count++;
          sum_num+=image.diff[h];
        }
    }
    for(uint16_t i=0; i<100; i++)
    {
        if(most_num[i]>max_tmp) {nor_num=i;max_tmp=most_num[i]/3;}
    }
    mean_num=sum_num/diff_count;
    uint16_t zebra_count=0,cross_count=0,zebra_flag=0,cross_flag=0,straight_
count=0,curve_count=0;
    uint16_t start_flag=0,start_count=0,stra2_count=0;
    for(uint16_t h=3; h<MT9V034_IMAGEH2-3; h++)
    {
        if(image.row[h]>35 && h>21)      {start_count++;}
        else if(image.row[h]<15) {start_count=0;}
        if(image.row[h]>60)      {zebra_count++;}
        else if(image.row[h]<40) {zebra_count=0;}
        if(image.row[h]>70)      {cross_count++;}
        else if(image.row[h]<40) {cross_count=0;}
        if(zebra_count>10) zebra_flag=1;
        if(cross_count>3) cross_flag=1;
        if(start_count>3) start_flag=1;
        if(abs(image.diff[h])<3)     {straight_count++;}
        if(image.row[h]>30 && h<24)  {curve_count++;}
        for(uint16_t w=5; w<MT9V034_IMAGEW2-5; w++)
```

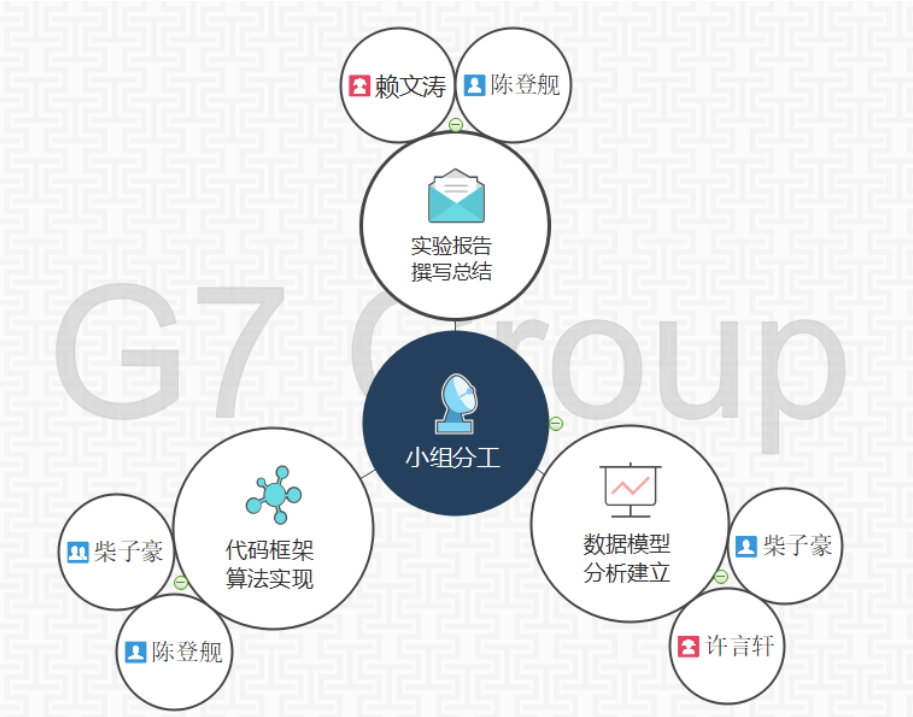
```
75.         {
76.            if(binary[h][w]==0 && abs(w-
    image.meanx)<3) {stra2_count++;break;}
77.         }
78.     }
79.     if(zebra_flag==1)
80.     {
81.         road_id=6;
82.         speed=0;
83.         dir=0;
84.         BOARD_I2C_GPIO(1<<(int)(8-dir*8));
85.     }
86.     else if(cross_flag==1)
87.     {
88.         road_id=4;
89.         speed=2;
90.         dir=0;
91.         BOARD_I2C_GPIO(1<<(int)(8-dir*8));
92.     }
93.     else if(start_flag==1)
94.     {
95.         road_id=5;
96.         speed=4;
97.         dir=0;
98.         BOARD_I2C_GPIO(1<<(int)(8-dir*8));
99.     }
100.     else if(curve_count>4 && straight_count>MT9V034_IMAGEH2/3)
101.     {
102.         road_id=3;
103.         dir= -(float)(tmp_line[15]-MT9V034_IMAGEW2/2)/MT9V034_IMAGEW2;
104.         speed = MINMAX(1,5-4*abs(dir),5);
105.         dir= MIN(MAX(dir, -0.99), 0.99);
106.         BOARD_I2C_GPIO(1<<(int)(8-dir*8));
107.     }
108.     else if(stra2_count>MT9V034_IMAGEH2*2/3)
109.     {
110.         road_id=1;
111.         dir = 0;
112.         speed =5;
113.         BOARD_I2C_GPIO(1<<(int)(8-dir*8));
114.     }
115.     else if(straight_count>MT9V034_IMAGEH2/2)
116.     {
```

```
117.          road_id=2;
118.          dir = -(float)(tmp_line[15]-MT9V034_IMAGEW2/2)/MT9V034_IMAGEW2;
119.          dir= MIN(MAX(dir, -0.99), 0.99);
120.          speed =5;
121.          BOARD_I2C_GPIO(1<<(int)(8-dir*8));
122.      }
123.      else
124.      {
125.          LED1_TOGGLE();
126.          road_id=0;
127.          speed = 6;
128.          dir = -1;
129.          BOARD_I2C_GPIO((1<<16)-1);
130.      }
131.      LED1_flash(121- speed * 20);
132. }
```

5. 实验结果分析与结论

本次实验以面阵CCD传感器信号检测为基础，通过对传感器输出信号的特征进行特征提取，进而根据车道线图像不同的信号特征，从而识别不同的路线，进而使对于小车的控制成为可能。

本次实验基本分工如下所示。



本次实验的重点在于对图像的处理及分析利用上。首先我们使用了高斯滤波进行预处理。二值化处理上，我们组尝试了大津法和平均值两种方法动态的求阈值，测试发现两种方法的实验结果都足以满足需求，但在图像边缘有较大的噪声问题，因此我们最终又融合了差分算子作为图像处理算法。在使用差分和二值化图像融合之后，我们使用了常用的腐蚀和膨胀方法去噪，得到的最终效果足以满足实验寻线需求。

在进行图像边缘提取的时候，我们尝试使用目前在边缘提取方面广泛使用的canny算法，但是最后发现效果并不理想，可能是由于图像噪点过多的原因。

我们还尝试了寻线算法，主要使用DFS和上一次在线阵CCD实验中使用的打擂计分寻线法。其中DFS算法基本可以完成寻线要求，可以追踪到最长的线段。

经过本次实验，我们基本熟悉示例代码Ref6的基本思路与代码实现方法，并以此代码为基础进行调整与改进，进行基本库函数的完善与封装，并整合加入了之前历次实验中使用的积累的各种算法，在此基础上完成了此次实验。

最后，感谢王冰老师为本门课程的精心准备与悉心讲解，深入浅出的为我们讲解嵌入式开发的诸多知识，将复杂的嵌入式知识网络按照对应模块分割开来，逐一细解，帮助我们拓宽视野，增长见识。