工程实践与科技创新3F 技术文档

巡线

巡线以霍夫变换直线检测为主体，其主要步骤如下：

1. 对摄像头捕获的图片依次进行透视变换、灰度化、高斯滤波、二值化、高斯滤波、Canny边缘检测
2. 对处理后的图片截取一段感兴趣区域
3. 对此区域进行霍夫变换，检测到水平和竖直的直线
4. 通过判定检测到的直线的斜率大小，剔除水平方向的直线
5. 通过对竖直方向的直线进行操作，确定出两个预瞄点
6. 使小车沿圆弧轨迹向预瞄点的中心行驶

其主体代码如下：

```
while (1)
{
    capture >> frame;

    if(frame_counter == 5)
    {
        // processing and control
    }
    else
    {
        frame_counter++;
    }
}
```

我们发现，对视频流的每一张图片均进行处理，会导致图片出现问题，进而引起后期处理问题。因此，我们采取每5张图片中处理一张并作出决策的措施。

```
warpPerspective(frame, color_img, mapperspective, Size(img_width,
img_height));

cvtColor(color_img, gray_img, CV_BGR2GRAY);

GaussianBlur(gray_img, gray_img, Size(7, 7), 1.5, 1.5);

threshold(gray_img, threshold_img, 110, 255, THRESH_BINARY_INV);
```

```
GaussianBlur(threshold_img, threshold_img, Size(7, 7), 1.5, 1.5);

Canny(threshold_img, threshold_img, 0, 30, 3);

refined_threshold_img = threshold_img(Rect(img_width/5, img_height/2+10,
img_width/2+10, img_height/3));

refined_color_img = color_img(Rect(img_width/5, img_height/2+10,
img_width/2+10, img_height/3));
```

图像预处理采用感兴趣区域透视变换&灰度化&二值化&边缘提取的方式。代码中，`frame`为视频流原图、`color_img`为透视变换之后的3通道彩色图片、`gray_img`为灰度化之后的单通道图片、`threshold_img`为二值化之后的图片。透视变换将图片变换为近似俯视图，使道路的识别更精确；高斯滤波的目的是使边缘检测的效果更好。

```
HoughLinesP(input_img, hough_line_set, 1, CV_PI / 180, 60, 30, 20);

for (line_iterator = hough_line_set.begin(); line_iterator !=
hough_line_set.end();)
{
    hough_line = *line_iterator;

    if ((hough_line[2] - hough_line[0]) == 0)
        line(output_img, Point(hough_line[0], hough_line[1]),
Point(hough_line[2], hough_line[3]), Scalar(12, 128, 255), 5, CV_AA);

    else
    {
        line_slope = (hough_line[3] - hough_line[1]) / (hough_line[2] -
hough_line[0]);

        if (line_slope * line_slope > 0.5)
            line(output_img, Point(hough_line[0], hough_line[1]),
Point(hough_line[2], hough_line[3]), Scalar(12, 128, 255), 5, CV_AA);

        else
        {
            line_iterator = hough_line_set.erase(line_iterator);
            line_iterator--;
        }
    }

    line_iterator++;
}
```

霍夫变换直线检测主体部分。霍夫变换会检测水平、竖直等所有符合条件的直线。通过直线斜率的判断，筛选出竖直的直线（斜率平方大于0.5），删除不符合条件的直线。代码中，`line_slope`为检测到每一条直线的斜率。

```
// in the loop
if(max_y_value > i * line_distance && min_y_value < i * line_distance)
{
    crosspoint_counter++;

    crosspoint[1] = i * line_distance;
    if(hough_line_set[j][1] != hough_line_set[j][3])
        crosspoint[0] = hough_line_set[j][2] - (hough_line_set[j][2]-
hough_line_set[j][0])*(i*line_distance-hough_line_set[j]
[3])/(hough_line_set[j][1]-hough_line_set[j][3]);
    else
        crosspoint[0] = (hough_line_set[j][0]+hough_line_set[j][2])/2;

    midpoint[0] = (midpoint[0]*(crosspoint_counter-
1)+crosspoint[0])/crosspoint_counter;
    midpoint[1] = (midpoint[1]*(crosspoint_counter-
1)+crosspoint[1])/crosspoint_counter;
}

if(i<=sample_num/2)
{
    up_point_counter++;
    setpoint[0] = setpoint[0] + midpoint[0];
    setpoint[1] = setpoint[1] + midpoint[1];
}
else
{
    down_point_counter++;
    setpoint[2] = setpoint[2] + midpoint[0];
    setpoint[3] = setpoint[3] + midpoint[1];
}

// after the loop
setpoint[0] = setpoint[0] / up_point_counter;
setpoint[1] = setpoint[1] / up_point_counter;
setpoint[2] = setpoint[2] / down_point_counter;
setpoint[3] = setpoint[3] / down_point_counter;
```

预瞄点提取。从图中选出两个预瞄点，以进行决策的制定。主要步骤如下：

1. 以一定数量（`sample_num`，程序中为10条）的水平直线与霍夫变换得到的竖直直线相交，得到很多交点。一般地，如果道路情况较好，两条边缘各检测出一条直线，则图中会有20个交点。
2. 对于每一条水平直线，将其与霍夫变换直线相交得到的所有交点取平均值，得到这一水平位置上道路的朝向情况。对于任一直线，若其与至少一条霍夫变换直线相交，则在其位置上的平均值点一定存在。
3. 将所有平均值点分为图片上方的一组和图片下方的一组。分别对两组点取平均值，得到两个预瞄点。若图片上方或下方没有平均值点，则返回一个点；若图片上没有平均值点，则不返回点。

代码中，`crosspoint`为步骤1中的交点、`midpoint`为步骤2中的平均值点、`setpoint`为步骤3中的预瞄点。

```
if(line_setpoint[1] == 1 && line_setpoint[0] != 0)
{
    angle = (line_setpoint[2]-base_point[0])/(line_setpoint[3]-
base_point[1]-0.1);
    steer = constrain(0.3*angle);

    if(abs(angle)>1.5)
    {
        usleep(2500000);
        speed=0.07;
    }
    else
    {
        speed=0.1;
    }

    pre_steer = steer;
    pre_speed = speed;
}
```

只有一个预瞄点的决策。line_setpoint为预瞄点提取函数的返回值，其第[1]个元素记录了预瞄点的数量，其第[2、3]个元素记录了预瞄点的坐标。base_point为图片底边的中点。决策中的速度为预设值，转角与图片底边中点&预瞄点的夹角成线性关系。

```
else if(line_setpoint[1] == 2 && line_setpoint[0] != 0)
{
    // sort the two setpoints
    if(line_setpoint[5] < line_setpoint[3])
    {
        tmp[0] = line_setpoint[2];
        tmp[1] = line_setpoint[3];
        line_setpoint[2] = line_setpoint[4];
        line_setpoint[3] = line_setpoint[5];
        line_setpoint[2] = tmp[0];
        line_setpoint[3] = tmp[1];
    }

    avgpoint[0] = (0.6*line_setpoint[2]+1.4*line_setpoint[4])/2;
    avgpoint[1] = (0.6*line_setpoint[3]+1.4*line_setpoint[5])/2;
    circle(img, Point(avgpoint[0], avgpoint[1]), 5, Scalar(0, 255, 0), -1,
8, 0);
    angle = (avgpoint[0]-base_point[0])/(avgpoint[1]-base_point[1]-0.1);

    steer = constrain(0.3*angle);
    if(abs(angle) > 1.5)
    {
        speed=0.07;
    }
    else
    {
```

```
            speed=0.1;
        }

        pre_steer = steer;
        pre_speed = speed;
    }
```

有两个预瞄点的决策。line_setpoint的第[2、3]、[4、5]个元素记录了两个预瞄点的坐标。程序先将预瞄点按从上到下排序，之后取两个预瞄点的加权平均值（上方预瞄点权值0.3，下方预瞄点权值0.7）。决策中的速度为预设值，转角与图片底边中点&平均预瞄点的夹角成线性关系。

```
if(abs(pre_steer)<0.4)
    steer = pre_steer;
else
    steer = 0.9*pre_steer/abs(pre_steer);
speed=pre_speed;
```

没有预瞄点的决策。没有预瞄点的情况基本为视频流图片中没有车道的情况。此时，按照前一次的决策制定当前决策：

- 若前一次决策转弯角度较大，证明小车已经进入弯道。此时认为小车已经驶入急弯，使小车转向值为0.9，方向与前一次决策的转向方向相同。
- 若前一次决策转弯角度较小，证明小车还在直线上。此时认为小车偶然没有识别出直线，使小车按照上一次决策的速度和转向值行驶。

返回

停车标志位识别与处理

巡线以霍夫变换圆检测为主体，其主要步骤如下：

1. 对处理后的图片截取一段感兴趣区域
2. 对摄像头捕获的图片依次进行灰度化、二值化、高斯滤波、Canny边缘检测
3. 对此区域进行霍夫变换，检测到圆
4. 通过判定检测到的圆的半径，判断小车与停车标志位的距离
5. 若检测到圆半径大于一定值（35个像素），则使小车停车

其主体代码如下：

```
cvtColor(parking_frame, gray_img, CV_BGR2GRAY);
threshold(gray_img, threshold_img, 200, 255, THRESH_BINARY);
GaussianBlur(threshold_img, threshold_img, Size(7, 7), 1.5, 1.5);
Canny(threshold_img, threshold_img, 0, 30, 3);
```

图像处理仍采取每5张图片中处理一张并作出决策的措施。 图像预处理采用与巡线近似的感兴趣区域灰度化&二值化&边缘提取的方式。高斯滤波的目的是使边缘检测的效果更好。

```
HoughCircles(threshold_img, circles, CV_HOUGH_GRADIENT, 2, 200, 100, 100,
10, 50);

if(circles.size() != 0)
{
    for(size_t i = 0; i < circles.size(); i++)
    {
        int radius = cvRound(circles[i][2]);
        if(radius > 35)
        {
            isParkingSign = 1;
            cout<<"car parked!"<<endl;
            return 1;
        }
    }
}
```

霍夫变换圆检测主体部分。若检测到圆半径大于一定值（radius，程序中为35），则返回小车已检测到停车标志（isParkingSign = 1;）。

isParkingSign作为返回值，外层函数检测到其值为1时，使小车停车。

返回

倒车入库

倒车入库以霍夫变换直线检测为主体，其主要步骤如下：

1. 对摄像头捕获的图片依次进行畸变矫正、透视变换、灰度化、高斯滤波、二值化、Canny边缘检测
2. 对此区域进行霍夫变换，检测到直线，并通过这些直线确定车库的中心点
3. 将车库中心点拟合成一条直线
4. 通过判定拟合直线在图片中的斜率和位置，判断小车的角度和位置误差
5. 在不同情况下采取不同的控制措施，使小车倒入车库中

其中，不同情况的具体细节如下：

1. 初步距离调节：小车检测到停车标志位并停车后，进行初步距离调节，使其与车库的距离在一个合理范围内（程序中小车与车库的合理距离范围为[70, 100]）。
2. 初步横向偏差修正：小车先以一个较小的角度向后倒退，然后以一个较大的角度向前行进。通过两次运动在水平方向上位移的差值修正横向偏差。
3. 初步角度偏差修正：小车通过检测图片上拟合直线的斜率，判断小车的角度误差，进而修正其位置使直线斜率在一定范围内（程序中直线斜率小于0.1）。
4. 再次距离调节：小车调节其与车库的距离，为其倒车入库做准备。
5. 倒车入库：小车对横向偏差进行检测。若其符合要求，则采用经验调节方式，按照S型曲线倒车入库。

其主体代码如下：

```
cvInitUndistortMap(intrinsic, distortion, mapxi, mapyi);
```

畸变矫正函数。后置摄像头畸变较为严重，因此单独在倒车入库过程中加入矫正步骤。代码中，`intrinsic`为视频流图片，`distortion`为矫正后的图片。

```
HoughLinesP(inputpic, lines2, 1, CV_PI / 180, 120, 100, 10);

for (size_t i = 0; i < lines2.size(); i++)
{
    Vec4i l = lines2[i];
    Point2i loc;
    if ((l[2] - l[0]) == 0)
    {
        loc.x = (l[2] + l[0]) / 2;
        loc.y = (l[3] + l[1]) / 2;

        prelocation.push_back(loc);
        yline.push_back(loc.y);
        midpoint_num++;
    }
    else
    {
        k = (l[3] - l[1]) / (l[2] - l[0]);
        if (k*k > 0.5)
        {
            loc.x = (l[2] + l[0]) / 2;
            loc.y = (l[3] + l[1]) / 2;

            prelocation.push_back(loc);
            yline.push_back(loc.y);
            midpoint_num++;
        }
    }
}
```

车库中心点标定。通过霍夫变换检测并筛选出车库间隔的直线，并画出每条直线的中点。代码中，`loc`为每一条车库间隔线的中点。

```
// linear fitting
fitLine(location, fit_line, CV_DIST_L2, 0, 0.01, 0.01);

point0.x = fit_line[2];
point0.y = fit_line[3];
kline = fit_line[1] / fit_line[0];

sort(xline.begin(), xline.end());
realxline.push_back(xline[0]);

for (int i = 0; i < ny - 1; i++)
{
    distance = xline[i + 1] - xline[i];
    if (distance > 30)
```

```
        {
            realxline.push_back(xline[i + 1]);
            realn++;
        }
    }

    turnloc.push_back(Point(realn, kline * 10000));

    for (int i = 0; i < realn; i++)
    {
        double parkx = (realxline[i] + realxline[i + 1]) / 2;
        double parky = kline * (parkx - point0.x) + point0.y;
        turnloc.push_back(Point(parkx, parky));
    }

    return turnloc;
```

将得到的中心点分组取平均值，得到每个车库的中心点。代码中，`realxline`为提取之后的车库间隔线，`turnloc`为标定的车库中心位置，其第[0]个元素为成功标定的车库中心数，当其值为999或0时，表示未成功标定任何一个车库。

```
if(step == 1)
    // distance adjustment
else if(step == 2)
    // horizontal error correction
else if(step == 3)
    // yaw angle error correction
else if(step == 4)
    // horizontal error correction
else if(step == 0)
    // car parking
```

在不同的情况中，对小车采用不同的控制措施。