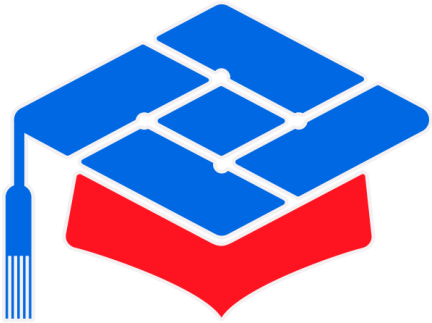


第八届

全国大学生集成电路创新创业大赛

报告类型：设计报告

参赛杯赛：海云捷迅

作品名称：基于 FPGA 的机械臂精确控制系统设计

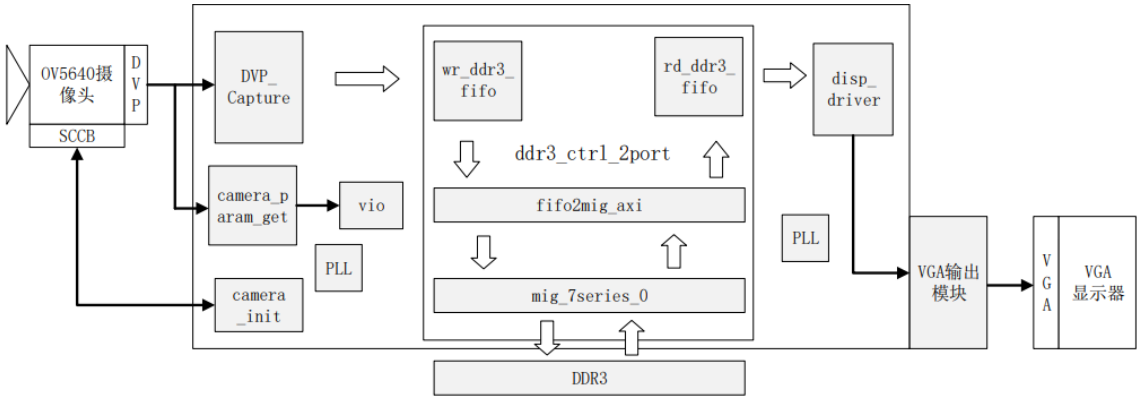
队伍编号：CICC2080

团队名称：困了就睡

图像检测部分

1.视频图像采集

视频图像采集模块使用 ov5640 摄像头采集，VGA 显示屏显示。其系统框图如图所示：



2.图像处理及分析

读 FIFO 输出数据先进入此模块，后将处理图像以及图像信息输出。

2.1 图像处理

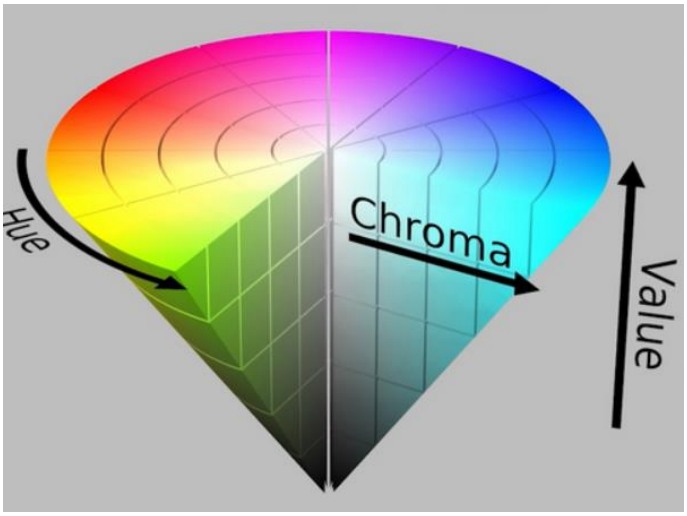
2.1.1 RGB 色域转 HSV 色域

HSV（色调，饱和度，亮度）使用色调区分颜色且相比 RGB 不易受光照影响，故选择将图像从 RGB 色域转换为 HSV 色域以进行后续处理，具体算法如下图所示。

$$h = \begin{cases} 0^\circ & \text{if } max = min \\ 60^\circ \times \frac{g-b}{max-min} + 0^\circ, & \text{if } max = r \text{ and } g \geq b \\ 60^\circ \times \frac{g-b}{max-min} + 360^\circ, & \text{if } max = r \text{ and } g < b \\ 60^\circ \times \frac{b-r}{max-min} + 120^\circ, & \text{if } max = g \\ 60^\circ \times \frac{r-g}{max-min} + 240^\circ, & \text{if } max = b \end{cases}$$

$$s = \begin{cases} 0, & \text{if } max = 0 \\ \frac{max-min}{max} = 1 - \frac{min}{max}, & \text{otherwise} \end{cases}$$

$$v = max$$



2.1.2 HSV 二值化处理

对色调，饱和度，亮度设置不同的阈值，对 HSV 信号进行过滤，满足阈值则显示白色，反之显示黑色。通过切换阈值即可实现不同颜色的识别同时得到颜色信息。以下是所测得不同颜色的阈值。

红色：(015,000,255,110,255,020)

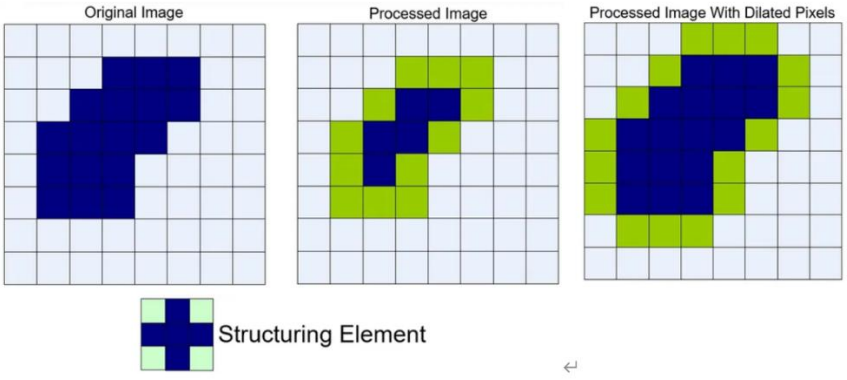
黄色：(175,000,255,000,045,000)

蓝色：(175,150,255,180,100,020)

黑色：(060,025,255,130,250,050)

2.1.2 开操作

对二值化的图像进行开操作，即先腐蚀后膨胀，消除画面中的噪点以得到稳定的二值化图像。具体为使用元素结构遍历整个图像，当某有效像素点作为元素中心不满足元素所示情况，则进行相应的腐蚀或膨胀操作。



2.2 图像分析

2.2.1 目标检测

遍历图像信息，将第一个有效像素点作为起始点，判断在其一定范围内是否有其他有效像素点，满足条件即视为一个整体并扩充其边界坐标。在一帧结束后输出其边界坐标。此部分参考了 b 站 up 主@大磊 FPGA 的图像处理思路。此方案的优点是相比使用连通域识别要节省很多资源。

```
always@(posedge Clk or negedge Rst_n)begin
    if(!Rst_n) begin
        location <= {1'b0,10'd0,10'd0,10'd0,10'd0};
        new_target_flag <= 'd0;
    end
    else if(vsync_neg_flag)begin    //在一帧开始进行初始化
        location <= {1'b0,10'd0,10'd0,10'd0,10'd0};
        new_target_flag <= 'd0;
    end
    else begin
        if(valid_i && img_data_i)begin
            if(target_flag == 1'b0)                 //数据无效
                location <= {1'b1,y_cnt,x_cnt,y_cnt,x_cnt};
            else begin                              //数据有效，则判断当前像素是否落在该元素临域里
                if((x_cnt < target_left)||(x_cnt > target_right)||(y_cnt < target_top)||(y_cnt > target_bottom))
                    new_target_flag <= 1'b0;        //如果坐标距离超出目标临域范围，投票认定为新的目标
                else
                    new_target_flag <= 1'b1;        //否则不认定为新的目标
            end
        end
        else if(valid_i_r && img_data_i_r) begin
            if(new_target_flag == 1'b1) begin       //未投票认定为新目标的元素，表示当前像素位于它的临域内
                location[40] <= 1'b1;
                if(x_cnt_r < location[ 9: 0] && !limit_x)       //若x坐标小于左边界，则将其x坐标扩展为左边界
                    location[ 9: 0] <= x_cnt_r ;
                if(x_cnt_r > location[29:20] && !limit_x)       //若x坐标大于右边界，则将其x坐标扩展为右边界
                    location[29:20] <= x_cnt_r ;
                if(y_cnt_r < location[19:10] && !limit_y)       //若y坐标小于上边界，则将其y坐标扩展为上边界
                    location[19:10] <= y_cnt_r ;
                if(y_cnt_r > location[39:30] && !limit_y)       //若y坐标大于下边界，则将其y坐标扩展为下边界
                    location[39:30] <= y_cnt_r ;
            end
        end
    end
end
```

2.2.2 形状识别

目标检测得到目标的范围后，下一帧记录此范围内有效像素点个数，三个形状差异明显。

```
always@(posedge Clk or negedge Rst_n)begin
    if(!Rst_n)begin
        Shape <= 3'b000;
        Shape_r <= 3'b000;
    end
    else if(update_r[0])begin
        if(pixel_cnt > 13800)//正方形
            Shape <= 3'b010;
        else if(pixel_cnt >= 10800 && pixel_cnt <= 13800)//圆形
            Shape <= 3'b100;
        else if(pixel_cnt < 10800)//六边形
            Shape <= 3'b001;
    end
    else if(update_r[3])
        Shape_r <= Shape;
end
```

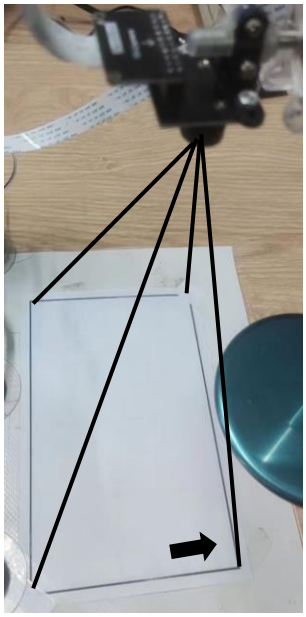
2.2.3 坐标转换

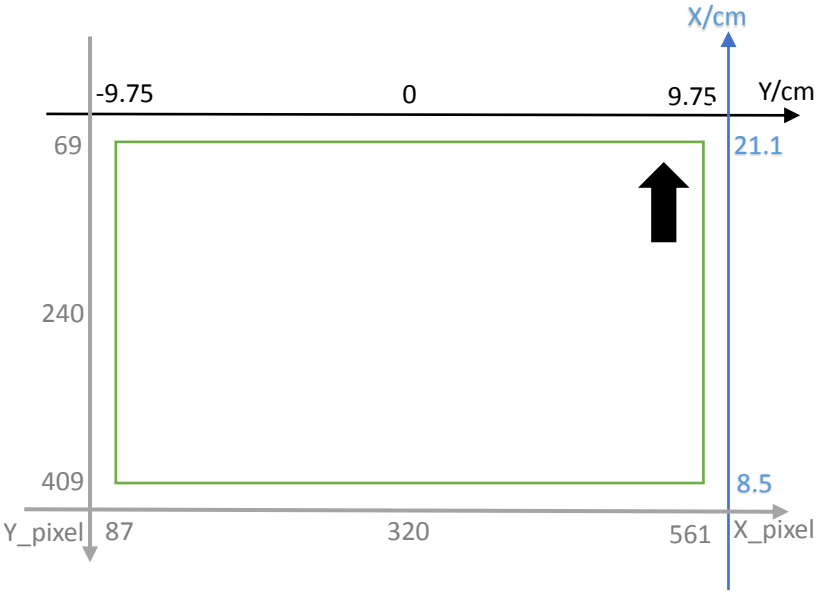
通过目标检测得到目标上下左右边界的坐标，显然：

Y_pixel = （上边界坐标 + 下边界坐标）>>1

X_pixel = （左边界坐标 + 右边界坐标）>>1

可得目标的中心像素坐标。代码中划分了有效区域用于对齐场地方框，实际坐标与像素坐标的关系如图所示：





根据二者关系可得转换公式为：

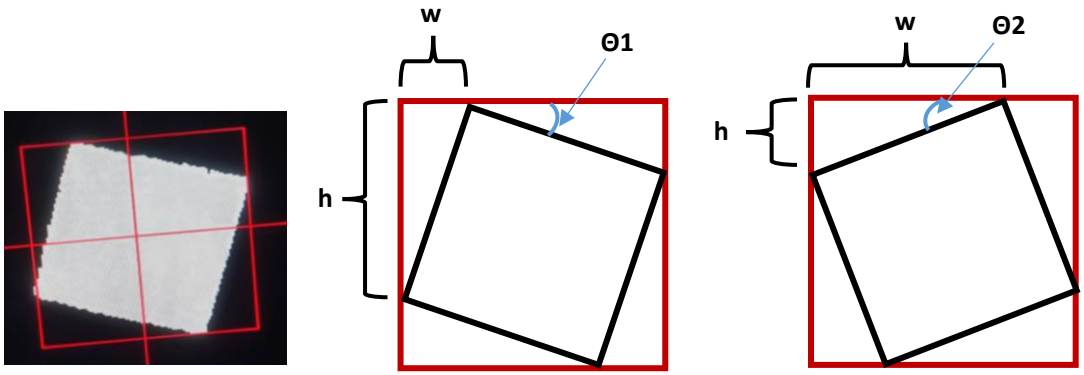
Y=(x_pixel-319)/0.241666666666666666666666666666667

X=(y_pixel-397)/0.259836065573770491803278688524 59 – 8.5

```
assign Y_pixel = update?((location_o[19:10] + location_o[39:30])>>1) : Y_pixel;
assign X_pixel = update?((location_o[9:0] + location_o[29:20])>>1) : X_pixel;
assign X_b = (((Y_pixel-397)*25222)>>16)-86;
assign Y_b = ((X_pixel-319)*27118)>>16;
```

2.2.4 正方形偏转角获取

当上一帧检测到目标为正方形，以上边界和左边界的交点作为起始点，分别从横向和纵向计数起始点到第一个有效点的长度，即可得到正方形的偏转角。具体原理如图所示：



当 h>w:

$$\Theta = \arctan(w/h)$$

当 h<=w:

$$\Theta = -\arctan(h/w)$$

分以上两种情况，令顺时针的偏移为正方向。由此可获得正方形的偏转角，其范围为-45°到 45°。其中 atan 的运算采用了查找表，相比 ip 核大大减少了资源占用。

2.2.5 抗粘连——视觉部分

当遍历所有颜色后（20 帧）未检测到有效目标，且检测到无效目标的有效像素数目远大于正方形的阈值，则判断为存在粘连目标，开启抗粘连模式。

抗粘连模式下，会限制目标范围的大小，从粘连的目标中准确选出一个移到远处，并重新识别。效果及部分代码如图所示：

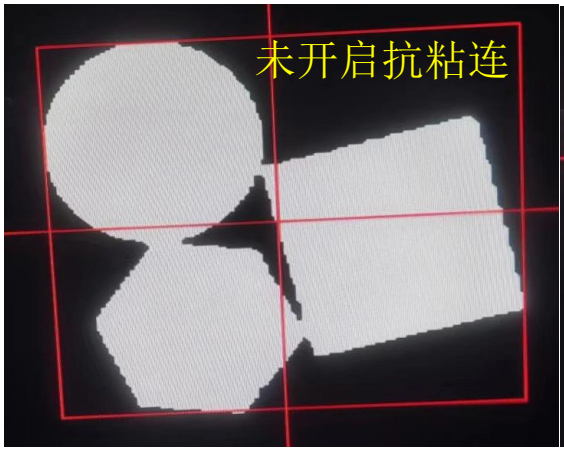
```
//////////////////////////////////////////判断粘连
    reg [3:0] cnt_aaa;//四个颜色遍历1遍共20帧，没有有效目标，则视为剩下的为粘连目标。
//////////////////////////////////////
    always@(posedge Clk or negedge Rst_n)begin
        if(!Rst_n)
            cnt_aaa <= 'b0;
        else if(No_goal && Data_req_r[0] && pixel_cnt >16000)
            cnt_aaa <= cnt_aaa + 1'b1;
        else if(cnt_aaa == 'd4 || Finish || Adhesion)
            cnt_aaa <= 'b0;
    end
    always@(posedge Clk or negedge Rst_n)begin
        if(!Rst_n)
            lock <= 'b0;
        else if(cnt_aaa == 'd4)
            lock <= 'b1;
        else if(Adhesion)
            lock <= 'b0;
    end
```

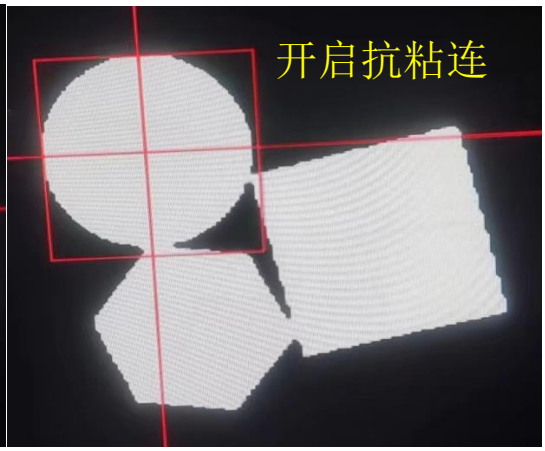
```
always@(posedge Clk or negedge Rst_n)begin
    if(!Rst_n)begin
        www <= 'b0;
        hhh <= 'b0;
    end
    else begin
        www <= location[29:20] - location[9:0];
        hhh <= location[39:30] - location[19:10];
    end
end
```

```
always@(posedge Clk or negedge Rst_n)begin
    if(!Rst_n)
        limit_y <= 'b0;
    else if(hhh >= 95 && lock)
        limit_y <= 'b1;
    else begin
        limit_y <= 'b0;
    end
end
```

```
always@(posedge Clk or negedge Rst_n)begin
    if(!Rst_n)
        limit_x <= 'b0;
    else if(www >= 85 && lock)
        limit_x <= 'b1;
    else
        limit_x <= 'b0;
end
```

```
if(x_cnt_r < location[ 9: 0] && !limit_x)
    location[ 9: 0] <= x_cnt_r ;
if(x_cnt_r > location[29:20] && !limit_x)
    location[29:20] <= x_cnt_r ;
if(y_cnt_r < location[19:10] && !limit_y)
    location[19:10] <= y_cnt_r ;
if(y_cnt_r > location[39:30] && !limit_y)
    location[39:30] <= y_cnt_r ;
```



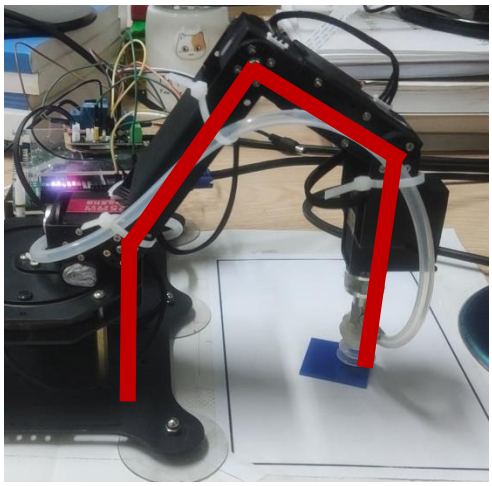


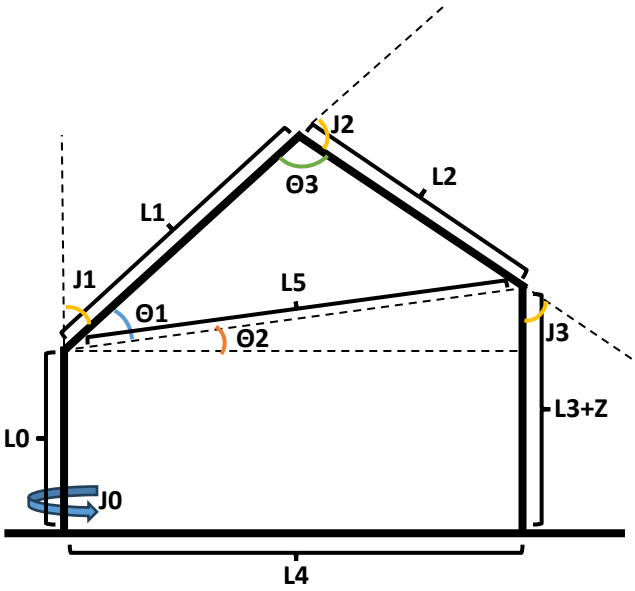
机械臂控制部分

1. 机械臂控制

1.1 逆运动学求解

忽略绕 Z 轴旋转的 0 号舵机，机械臂的姿态仅由 3 个舵机控制，末端的 3 号舵机只须保持垂直地面的姿态，由此可得以下方案：





其中 L0~L3 和坐标(X,Y,Z)已知,计算过程如下：

$$L_4 = \sqrt{x^2 + y^2},\ L_5 = \sqrt{(L_3 + z - L_0)^2 + L_4^2}$$

$$cos\,\theta_1 = \frac{L_1^2 + L_5^2 - L_2^2}{2L_1L_5},\ cos\,\theta_3 = \frac{L_1^2 + L_2^2 - L_5^2}{2L_1L_5}$$

$$\theta_1 = arccos(cos\,\theta_1),\ \theta_2 = arccos\left(\frac{L_4}{L_5}\right),\ \theta_3 = arccos(cos\,\theta_3)$$

$$j_0 = arccos\left(\frac{y}{L_4}\right),\ j_1 = 90° - \theta_1 - \theta_2,\ j_2 = 180° - \theta_3,\ j_3 = \theta_1 + \theta_2 + \theta_3 - 90°$$

整个模块只需使用两个 ip 核：DIV 和 sqrt。其中 arccos 的计算使用查找表实现。

1.2 J4 的计算（正方形角度保持的实现）

当机械臂到达起始点（目标坐标），记录此时的 J0_start，同时机械臂运行前已经得到目标信息，可知终点信息 J0_end 和正方形偏转角度 Deflection_angle。

则正方形到达终点后的总偏转角度为：

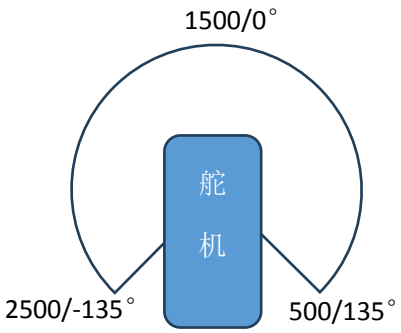
Angle_change = J0_end – J0_start - Deflection_angles

而正方形每偏转 90 度相当于没有变化，则当 Angle_changes 在不同的范围内，减去无意义的角度变化，即可得 J4。具体内容为：

```
if(angle_change <= 450)
    J4 <= angle_change;
else if(angle_change > 450 && angle_change <= 1350)
    J4 <= angle_change - 'd900;
else if(angle_change > 1350 && angle_change <= 2250)
    J4 <= angle_change - 'd1800;
else if(angle_change > 2250)
    J4 <= angle_change - 'd2700;
```

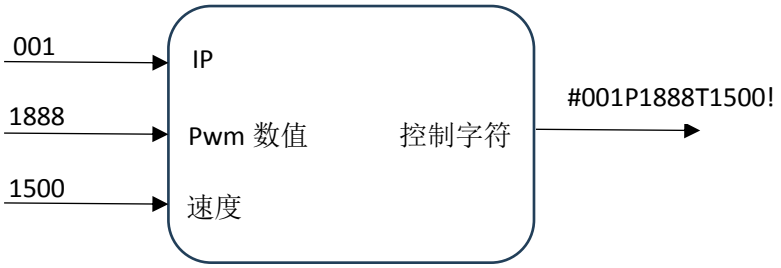
1.3 角度转 PWM

项目中舵机均使用 270 度顺时针模式，将 270 度分为 2000 个 PWM 单位，即每个单位代表 0.135 度。姿态解算得到每个舵机的角度后，根据该比例，将角度值转换成为对应的 PWM。



1.4 控制指令生成

获得每个舵机的 PWM 值后，生成控制字符，然后通过串口发送给机械臂。



2. 动作组

2.1 抓取动作

机械臂的控制为输入坐标（x，y，z）以及使能信号，末端即可到达坐标点。而动作组的实现则是通过状态机周期性的给机械臂控制模块不同的坐标以完成到达目标点、下探抓取和放置等动作。

默认状态下，机械臂会在待机位置，同时信号请求标志位为高电平。当信号请求标志位为高电平且图像分析模块的信息采集完成标志位为高电平，则记录此时传递的信息。并控制机械臂完成对应的动作。

```
case(step)
'd0:begin//发送
    EN <= 'b0;
    step <= (update_r2||move_finish_r||Anti_adhesion_r2)?step_r:0;
    Data_req_before <= (update || Anti_adhesion)?'b0:Data_req_before;
    J0_flag <= 'b0;
end
'd1:begin//发送
    EN <= 'b1;
    step <= 'd0;
end
'd2:begin//待机
    X <= 'd0;
    Y <= 'd150;
    Z <= 'd40;
    step <= 'd1;
    Data_req_before <= 'b1;
end
'd3:begin//到达缓冲点—始
    X <= -'d150;
    Y <= 'd0;
    Z <= 'd40;
    step <= 'd1;
end
'd4:begin//到达起始点
    X <= X_start_r;
    Y <= Y_start_r;
    Z <= 'd40;
    J0_flag <= J0_flag_r;
    step <= 'd1;
end
```

```
'd5:begin//下探和抓取
    X <= X_start_r;
    Y <= Y_start_r;
    Z <= 'd10;
    J0_flag <= J0_flag_r;
    step <= 'd1;
    Pump <= 'b1;
end
'd6:begin//抬起
    X <= X_start_r;
    Y <= Y_start_r;
    Z <= 'd40;
    step <= 'd1;
end
'd7:begin//到达目标点
    X <= X_target;
    Y <= Y_target;
    Z <= Z_target;
    step <= 'd1;
end
'd8:begin//放置和抬升
    Pump <= 'b0;
    step <= (cnt >= 'd13000000)?'d2:step;
end
```

```
Robotic_arm_control_ak Robotic_arm_control_ak(
    .Clk            (sys_clk),
    .Rst_n          (rst_n),

    .EN             (EN),
    .X              (X),
    .Y              (Y),
    .Z              (Z),

    .height_r       (height_r),
    .width_r        (width_r),
    .J0_flag        (J0_flag),
    .J0_end         (J0_end),
    .update         (update),

    .Uart_tx_s      (Uart_tx_s),//输出控制指令
    .move_finish    (move_finish) //移动完成标志位
);
```

2.2 抗粘连——机械臂部分

判断为粘连后返回抗粘连标志位，执行另一套动作组，将粘连目标中的一个移开。部分代码如下：

```
'd10:begin//到达缓冲点—始
    X <= -'d150;
    Y <= 'd0;
    Z <= 'd40;
    step <= 'd1;
end
'd11:begin//到达粘连中心
    X <= X_start_r;
    Y <= Y_start_r;
    Z <= 'd40;
    step <= 'd1;
end
'd12:begin//拨乱反正
    X <= X_start_r;
    Y <= Y_start_r;
    Z <= 'd10;
    step <= 'd1;
    Pump <= 'b1;
end
```

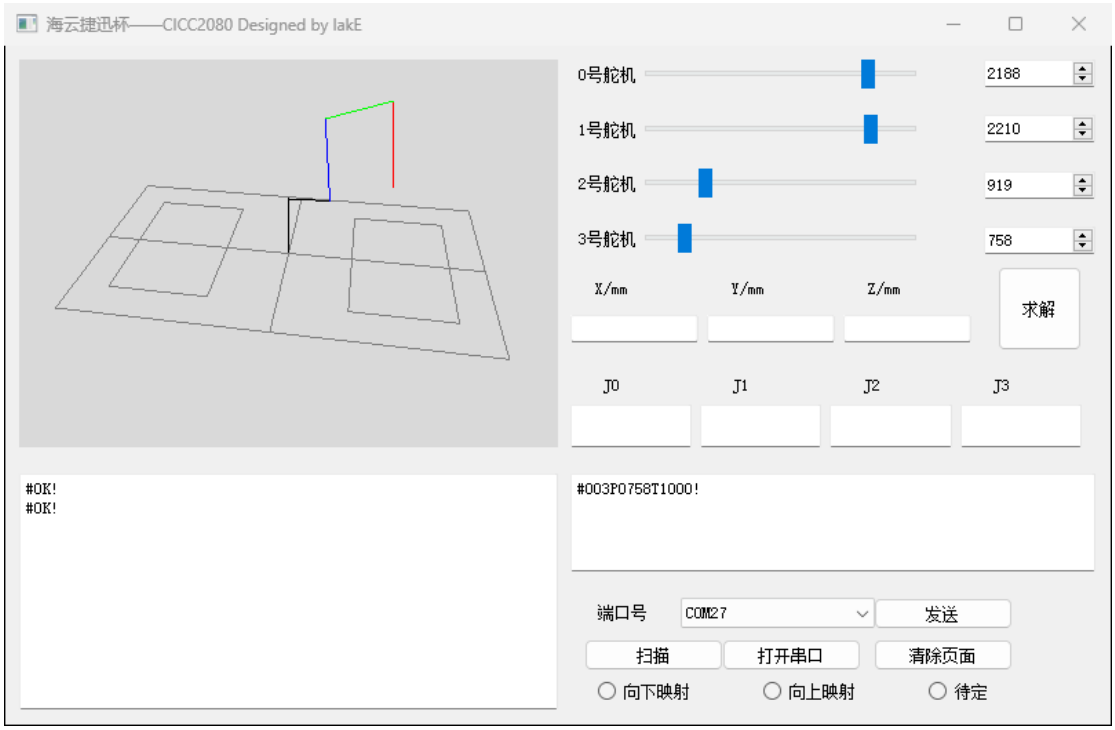
```
'd13:begin
    X <= X_start_r;
    Y <= Y_start_r;
    Z <= 'd40;
    step <= 'd1;
end
'd14:begin
    X <= X_correct;
    Y <= Y_correct;
    Z <= 'd40;
    step <= 'd1;
end
'd15:begin//放置和抬升
    Pump <= 'b0;
    step <= (cnt >= 'd13000000)?'d2:step;
end
default:step <= 'd0;
```

```
//////////////////////拨开粘连方向
    always@(posedge Clk or negedge Rst_n)begin
        if(!Rst_n)
            X_correct <= 'd0;
        else if($signed(X_start_r) < $signed(-145))
            X_correct <= -'d120;
        else
            X_correct <= -'d170;
    end
    always@(posedge Clk or negedge Rst_n)begin
        if(!Rst_n)
            Y_correct <= 'd0;
        else if(Y_start_r[15])
            Y_correct <= 'd50;
        else
            Y_correct <= -'d50;
    end
```

拓展部分

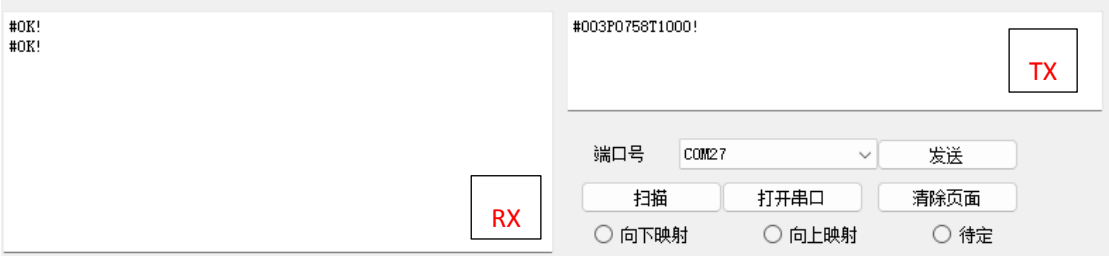
1. 上位机介绍

该上位机使用 Python 调用 QT 库制作完成，如图为上位机界面，下面将逐一讲解各部分功能。



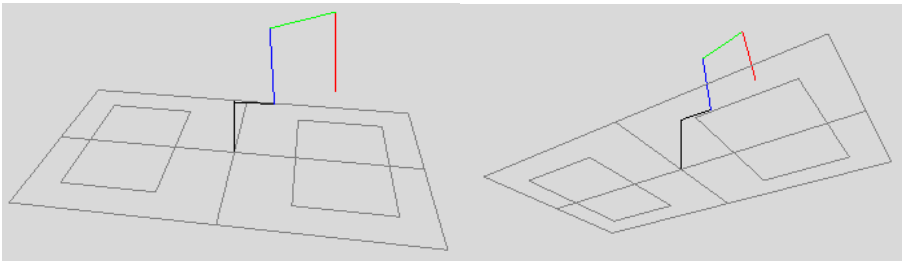
1.1 串口通信

该部分为串口通信模块，点击扫描自动搜寻可用串口并在下拉列表中查看选择，点击打开串口即可连接所选串口，该模块用于上位机和机械臂的通信，也可当作普通的串口助手。默认波特率为 115200。



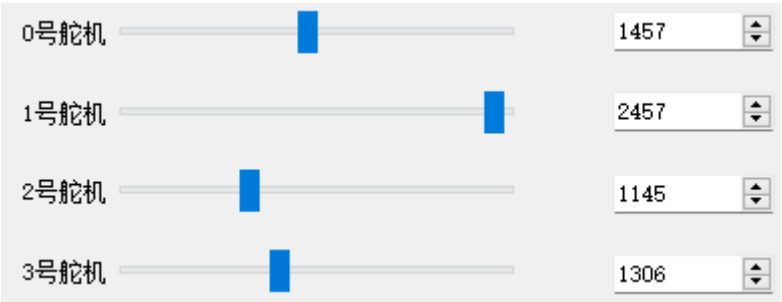
1.2 数字孪生

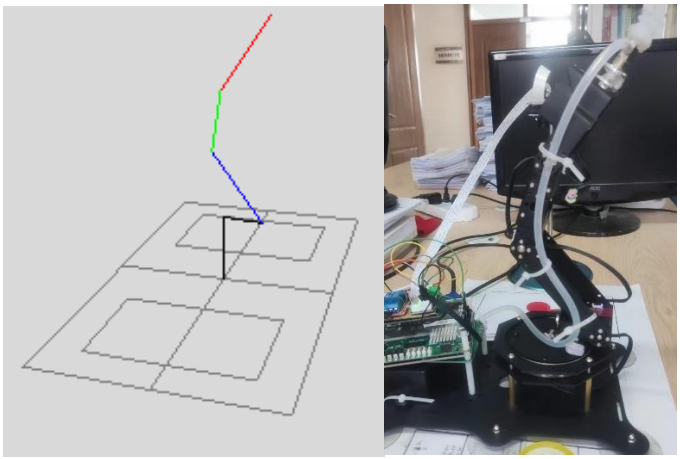
该部分为一个简易的机械臂模型，可以进行动作预览和动作同步。鼠标长按拖动可以任意变换视角。



1.3 舵机控制

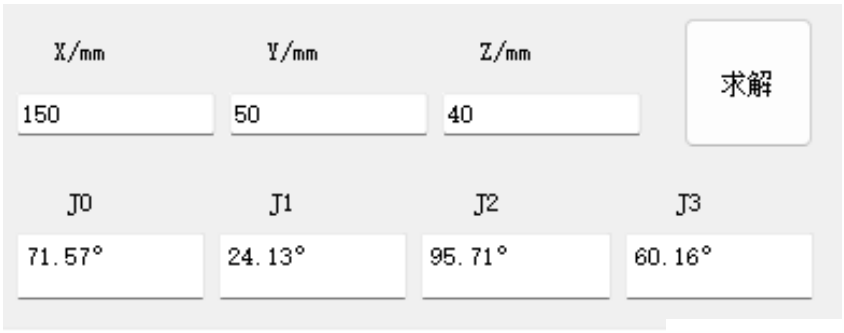
该部分为 4 个滑块和文本框，通过拖动滑块或者直接输入的方式，改变舵机的 PWM 值。数字机械臂会执行相应的动作，同时上位机会发送动作指令给机械臂。



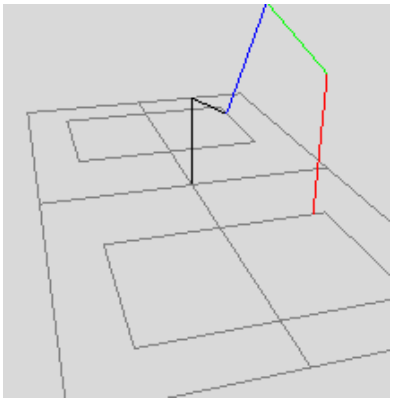


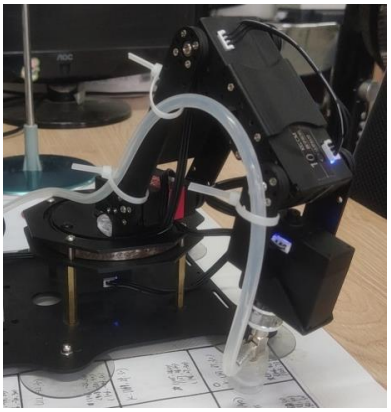
1.4 逆运动学求解

该部分为求逆解模块，输入末端坐标(X,Y,Z)（单位：毫米），点击求解即可得到各个舵机的偏转角度，数字机械臂会执行相应动作，同时上位机会发送动作指令给机械臂。



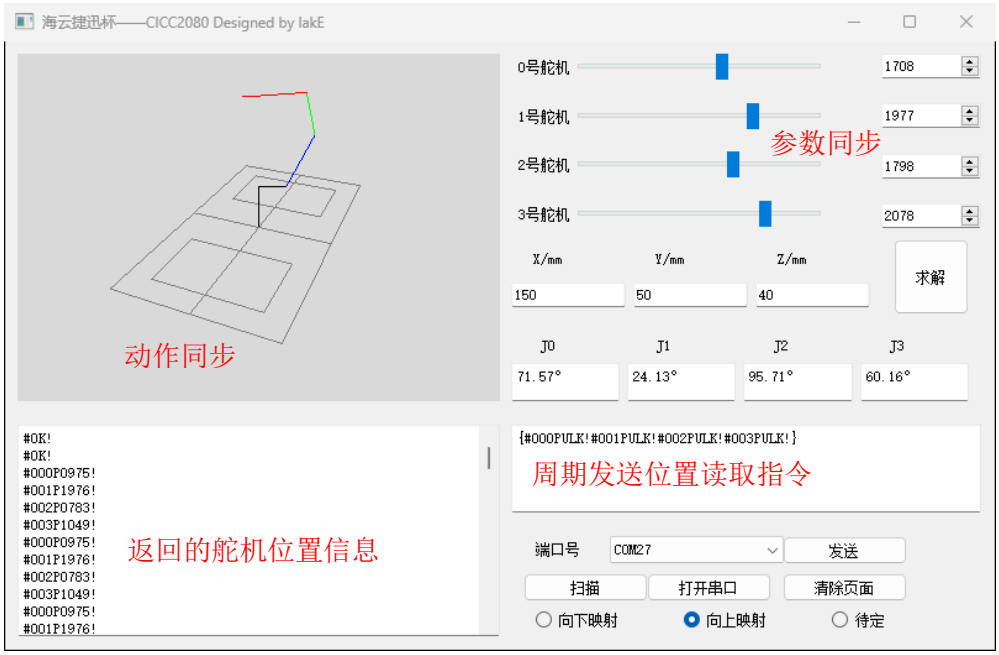
{#000P0969T1500!#001P1987T1500!#002P0791T1500!
#003P1054T1500!}





1.5 动作同步

点击向上映射，上位机会发送卸力指令并不断的读取每个舵机的位置，此时用手掰动机械臂，数字机械臂会同步其姿态动作。


海云捷迅杯——CICC2080 Designed by lakE
动作同步
0号舵机
1708
1号舵机
1977
参数同步
2号舵机
1798
3号舵机
2078
X/mm
Y/mm
Z/mm
150
50
40
求解
J0
J1
J2
J3
71.57°
24.13°
95.71°
60.16°
#OK!
#OK!
#000P0975!
#001P1976!
#002P0783!
#003P1049!
#000P0975!
#001P1976!
#002P0783!
#003P1049!
#000P0975!
#001P1976!
返回的舵机位置信息
{#000PULK!#001PULK!#002PULK!#003PULK!}
周期发送位置读取指令
端口号
COM27
发送
扫描
打开串口
清除页面
向下映射
向上映射
待定