拖拉拽流程编辑器图像处理小工具

BiCK

2024 年 9 月 20 日

目录

1 UI 界面创建

1.1 CSharp 工程创建

C# 界面开发使用 Windows 窗体应用 (.Net Framework)。

WinForms(Windows Forms) 是微软为.Net Framework 提供的图像用户界面 GUI 库，用于构建基于 Windows 平台的桌面应用程序。

图 1.1: 选择 winForms

1.2 UI 排版

使用 GroupBox 创建 UI 界面，并进行相应的名称更改，如图1.2所示：

图 1.2: 可视化界面

接着使用 Button、Label、Panel 控件将主要 UI 界面内容绘制出来。这里要注意的一点是像” 加载图像”、” 灰度图像” 等这些控件要选择 label 标签。还有一点要注意的是：像 label 和流程编辑的

panel，需要将”AllowDrop” 这个选项置为”True”。

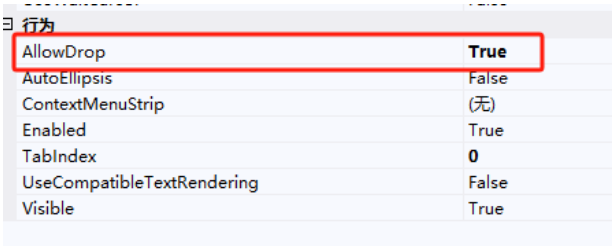

图 1.3: 行为里面的 AllowDrop 置为 True

图 1.4: 添加控件后的界面

1.3 移动控件自定义

因为要在流程编辑里面使用拖拉方式，所以需要自定义绘制控件。可以先创建一个文件夹，然后在文件夹里面右键添加” 用户控件 (Windows 窗体)”，然后绘制图像。

图 1.5

这里因为后面需要根据工具箱中的控件更改移动自定义控件上面的文字名称，需要创建一下控件的自定义属性：

```
1 using HalconDotNet;
2 using System;
3 using System.Collections.Generic;
```

```
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace halconCSharp2.VControl
{
    public partial class FlowNode : UserControl
    {

        public string NodeID;
        public string preNodeID;
        public string nextNodeID;

        // 定义输入与输出字段
        public HObject input;
        public HObject output;

        // 定义一个字段用于记录上一个节点数据
        private static FlowNode previousNode;

        // 创建控件的自定义属性
        [Category("流程控件属性")]
        public string NodeName { get => title.Text; set => title.Text = value; }
        [Category("流程控件属性")]
        public Color LightColor { get => label_LED.ForeColor; set => label_LED.
            ForeColor = value; }

        public FlowNode()
        {
            InitializeComponent();
            NodeID = Guid.NewGuid().ToString().Replace("-", "␣");
        }
    }
}
```

其中 get 和 set 访问器：

1. get 访问器，当外部代码读取 NodeName 属性时，实际返回的是 lblTitle.Text 的值，也就是 title 标签的文本,(这里的 title 是之前自定义控件右边 label 的 name)

2. set 访问器，当外部代码设置 NodeName 属性时，传入的值会被赋给 lblTitle.Text，从而改变 title 标签的显示文本

InitializeComponent()：windows forms 自动生成的初始化方法，通常用于设置控件的初始状态、布局、事件等。每个窗体或用户控件的构造函数都会调用这个方法，确保所有控件都已初始化并可用

Guid 是 C# 中的一个结构体，用于生成全局唯一标识符（GUID），NewGuid() 是 Guid 的静态方法，它返回一个新的、随机生成的 GUID。

Replace 方法用于将字符串中的某个字符或子字符串替换为指定的其他字符或字符串例子中，会将所有短横向替换为空格” ”；GUID 格式为:xxxxxxxx-xxxx-xxxx-xxxx-xxxxxxxxxxxx

1.4 工具箱显示隐藏

工具箱实现功能，按下按键隐藏和展开切换，如图1.6所示。

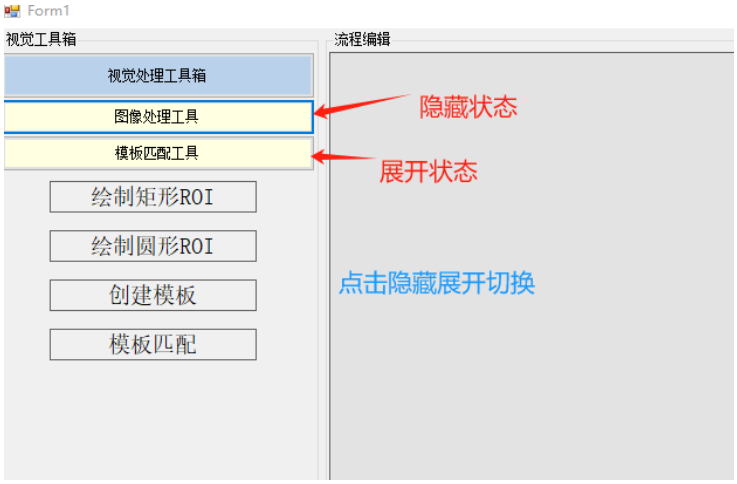

图 1.6: 展开和隐藏切换

这个的实现方式就是通过点击按钮，放置相关图标的 panel 的可见性 true 和 false 切换。

```
1  private void btn_visionTool_Click(object sender, EventArgs e)
2  {
3      panel_image.Visible = false;
4      panel_template.Visible = false;
5  }
6
7  private void btn_imageProcess_Click(object sender, EventArgs e)
8  {
9      panel_image.Visible = !panel_image.Visible;
10 }
11
12 private void button1_Click(object sender, EventArgs e)
13 {
14     panel_template.Visible = !panel_template.Visible;
15 }
```

2 拖拉拽实现

2.1 label 的拖拽

WinForms 拖拉拽实现思路：

1. 启动拖动操作
   - 当用户按下鼠标按钮时，通过控件的 MouseDown 事件启动拖动操作。
   - 在启动拖动时，可以通过 DoDragDrop() 方法来触发拖动行为。
2. 拖动过程中的视觉反馈
   - 拖动过程中，通过 DragEnter、DragOver 事件进行拖动视觉反馈更新，例如改变鼠标指针、显示影子控件等
3. 拖动释放并放置
   - 当用户将控件拖动到目标控件并释放鼠标按钮时，通过 DragDrop 事件完成拖放操作。
4. 处理鼠标移动和释放
   - 在 MouseMove 事件中，通过改变控件的位置来实现拖动效果，使用 MouseUp 事件完成拖动结束。

2.2 C# 的注册事件

在 C# 中，**注册事件** (或事件订阅) 是指将一个方法绑定到一个事件上，当事件触发时，绑定的方法将自动被调用。这是一种典型的**事件驱动编程**方式，常用于处理用户界面中的交互 (如按钮点击、鼠标移动等)。

2.2.1 事件注册的基本概念

1. **事件**：事件是一个委托类型的成员，用来定义某些动作 (如鼠标点击、键盘按下) 的处理机制。
2. **委托**：委托是一种类型安全的函数指针，它可以指向具有特定参数和返回类型的方法。
3. **事件处理器**：这是一个方法，包含事件发生时应执行的逻辑。事件处理器的签名必须与事件委托的签名一致。

2.2.2 C# 注册事件的方式

基本步骤：

1. **定义事件**：在类中定义事件，通常使用 event 关键字。
2. **触发事件**：在适当的时机触发事件 (通常在某个条件满足时)。
3. **订阅事件**：将处理该事件的方法注册到事件中。
4. **取消订阅事件**：可以使用`-=`从事件中取消方法注册。

2.2.3 事件注册示例

假如有一个按钮控件，想要在点击该按钮时执行某个方法，通过注册按钮的`Click`事件来实现。

```
Button myButton = new Button();

// 定义事件处理器
myButton.Click += new EventHandler(MyButton_Click);

// 事件处理器方法
void MyButton_Click(object sender, EventArgs e)
{
    System.Diagnostics.Debug.WriteLine("按钮被点击");     debug版本运行
    System.Diagnostics.Trace.WriteLine("按钮被点击");      debug和release都能运行
    MessageBox.Show("按钮被点击");
}
```

在这个例子中，myButton.Click 事件通过 += 操作符注册了一个事件处理器 MyButton_Click。当用户点击按钮时，MyButton_Click 方法会被自动调用

取消事件注册可以使用`-=`操作符取消事件的订阅：

```
myButton.Click -= new EventHandler(MyButton_Click);
```

2.3 显示面板的范围限制

在流程编辑界面上移动控件的时候，需要限制区域，不能让其移除限制区域。

实现代码：

```
// sender是触发事件的控件，as control将它转换为 control 类型的对象
var control = sender as Control;
control.Cursor = Cursors.Hand;
// .parent 用于获取父级控件或容器，
// 如果control是一个按钮，而这个按钮位于一个panel容器中，
// control.Parent就会返回这个Panel对象
var node = control.Parent;
// 获取控件的偏移量
int left = node.Left + (e.X - startPoint.X);
int top = node.Top + (e.Y - startPoint.Y);
// 获取控件本身的宽和高
int width = node.Width;
int height = node.Height;
// 动态获取流程编辑区域的大小
var rect = node.Parent.ClientRectangle;
// 判断拖拽的过程中不能超出边界
left = left < 0 ? 0 : ((left + width > rect.Width) ? rect.Width - width : left);
top = top < 0 ? 0 : ((top + height > rect.Height) ? rect.Height - height : top);

```

```
// 设置控件新的偏移量
node.Left = left;
node.Top = top;

// 强制刷新流程控件
// Invalidate() 是control类的一个方法，用于通知系统该控件的显示区域无效，并且需要重新绘制或刷新
panel1.Invalidate();
```

2.4 将控件从工具栏移动到流程编辑界面的代码

```
/// <summary>
/// 所有菜单对应鼠标左键按下事件
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void label_Tool_MouseDown(object sender, MouseEventArgs e)
{
    // 判断是否按下了鼠标左键
    if (e.Button == MouseButtons.Left)
    {
        // 这个参数通常代表启动拖动操作的对象，在此情况下，传递给 DoDragDrop 作为拖动的数据对象
        // sender 通常是在事件触发时的对象，比如你拖动的控件或元素，这个数据会被传递到目标控件的DrawEnter或DragDrop中
        lbl = sender as Label;

        // 设置当前被点击的工具上的鼠标样式
        // DragDropEffects.Copy: 表示拖放操作应该复制数据，也就是说，拖放完成后，
        // 原始数据保留在源控件中，同时将数据复制到目标控件中。
        // DragDropEffects.Move: 表示拖放操作应该移动数据，完成拖放后，数据会从源控件中删除，并放到目标控件中。
        lbl.DoDragDrop(sender, DragDropEffects.Copy | DragDropEffects.Move);    // DoDragDrop 启动拖放操作
    }
}

/// <summary>
/// 工具拖拽到流程编辑区时显示拖拽的鼠标样式效果
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void PL_FlowProcess_DrawEnter(object sender, DragEventArgs e)
{
    if (e.Data.GetDataPresent(typeof(Label)))
```

```
31      {
32          // Copy: 在拖放结束后，源控件的内容保持不变，数据被复制到目标控件。
33          // Move: 在拖放结束后，源控件的内容将被移除，数据移动到目标控件。
34          e.Effect = DragDropEffects.Copy | DragDropEffects.Move;
35      }
36  }
37
38  /// <summary>
39  /// 把工具拖放到编辑区域完成后方式
40  /// </summary>
41  /// <param name="sender"></param>
42  /// <param name="e"></param>
43  private void panel1_DragDrop(object sender, DragEventArgs e)
44  {
45      System.Diagnostics.Debug.WriteLine("panel1_DragDrop");
46      var container = sender as Control;
47      // 创建自定义流程对象
48      FlowNode flownode = new FlowNode();
49
50      // 给生成的新自定义控件对象注册对应鼠标事件
51      MouseEventHelper.RegistryMouseEvent(flownode, NodeMouseDown, MouseEventName.
            MouseDown);  // 鼠标按下
52      MouseEventHelper.RegistryMouseEvent(flownode, NodeMouseMove, MouseEventName.
            MouseMove);  // 鼠标移动
53      MouseEventHelper.RegistryMouseEvent(flownode, NodeMouseUp, MouseEventName.
            MouseUp);  // 鼠标松开
54      MouseEventHelper.RegistryMouseEvent(flownode, NodeMouseClick, MouseEventName.
            MouseDown);  // 鼠标点击
55
56      // 设置流程节点文本
57      flownode.NodeName = lbl.Text;
58
59      // 设置控件的位置
60      flownode.Location = container.PointToClient(new Point(e.X, e.Y));
61
62      // 把按钮添加到对应位置
63      container.Controls.Add(flownode);
64
65  }
66
67  /// <summary>
68  /// 在流程节点上按下鼠标左键处理事件
69  /// </summary>
70  /// <param name="sender"></param>
71  /// <param name="e"></param>
72  private void NodeMouseDown(object sender, MouseEventArgs e)
```

```
{
    isMoving = true;
    startPoint = new Point(e.X, e.Y);
}

/// <summary>
/// 鼠标在流程节点上松开的事件
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void NodeMouseUp(object sender, MouseEventArgs e)
{
    // 鼠标移动结束
    if (isMoving == true)
    {
        isMoving = false;
    }
}

/// <summary>
/// 鼠标按下在控件上移动过程进行的设置操作事件
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void NodeMouseMove(object sender, MouseEventArgs e)
{
    if (isMoving)
    {
        if (e.Button == MouseButtons.Left)
        {
            // sender是触发事件的控件，as control将它转换为 control 类型的对象
            var control = sender as Control;
            control.Cursor = Cursors.Hand;
            // .parent 用于获取父级控件或容器，
            // 如果control是一个按钮，而这个按钮位于一个panel容器中，
            // control.Parent就会返回这个Panel对象
            var node = control.Parent;
            // 获取控件的偏移量
            int left = node.Left + (e.X - startPoint.X);
            int top = node.Top + (e.Y - startPoint.Y);
            // 获取控件本身的宽和高
            int width = node.Width;
            int height = node.Height;
            // 动态获取流程编辑区域的大小
            var rect = node.Parent.ClientRectangle;
            // 判断拖拽的过程中不能超出边界
```

```
119             left = left < 0 ? 0 : ((left + width > rect.Width) ? rect.Width - width
                    : left);
120             top = top < 0 ? 0 : ((top + height > rect.Height) ? rect.Height - height
                     : top);
121
122             // 设置控件新的偏移量
123             node.Left = left;
124             node.Top = top;
125
126             // 强制刷新流程控件
127             // Invalidate() 是control类的一个方法，用于通知系统该控件的显示区域无
                    效，并且需要重新绘制或刷新
128             panel1.Invalidate();
129         }
130     }
131 }
```

2.5 连接线处理

2.5.1 右击控件弹窗

选择控件”ContextMenuStrip”，因为在流程编辑窗口右键触发这个右键菜单，所以需要在流程编辑窗口绑定一下，在行为里面的 ContextMenuStrip 选择这个右键菜单窗口。

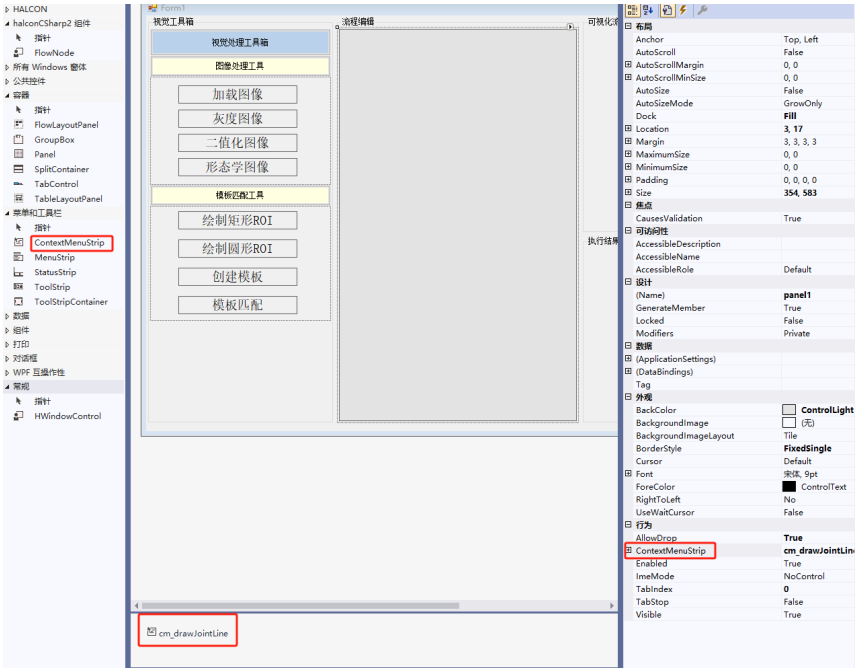

图 2.1: 右键菜单

这个点击右键菜单主要是为了进入连线状态。

2.6 选择连线的节点

这里主要也是通过之前注册的流程控件的事件来处理，判断前后两次点击的 FlowNode 编号，先点击的为第一个，后点击的为第二个，差不多是链表的一个思想。

```
/// <summary>
/// 点击流程控件的事件,用来连线
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void NodeMouseClick(object sender, MouseEventArgs e)
{
    if (e.Button == MouseButtons.Left)
    {
        // 判断是否为连线状态
        if (drawState == DrawState.DrawLine)
        {
            var control = sender as Control;
            if (nodeNum == 0) // 开始节点
            {
                if (control.Parent != null)
                {
                    // 获取连接点的一个节点
                    node1 = (FlowNode)control.Parent;
                    nodeNum = 1;
                }
                else
                {
                    MessageBox.Show("请选择一个流程节点");
                }

            }
            else if (nodeNum == 1)
            {
                // 绘制两个节点连线操作
                if (control.Parent != null)
                {
                    node2 = (FlowNode)control.Parent;
                    // 判断点击是否为同一个节点
                    if (!node2.Equals(node1))
                    {
                        // 设置从属关系
                        node1.nextNodeID = node2.NodeID;
                        node2.preNodeID = node1.NodeID;

                        // 找到了第二个节点，需要把nodeNum重置为0，为下一次连线做准
                            备
```

```
                        nodeNum = 0;
                        // 绘制连线
                        DrawPointToPointLine(node1, node2);
                        // 恢复正常绘制状态
                        drawState = DrawState.Normal;
                    }
                }
                else
                {
                        MessageBox.Show("请选择需要连线目标节点");
                }
            }
        }
    }
}
```

2.6.1 连接线的实现方式

连接线的方式主要是判断” 从左到右”、” 从右到左”、” 从上到下”、” 从下到上” 这几种方式。这个判断的方法就是依据两个节点 x 方向和 y 方向的距离信息。

```
/// <summary>
/// 绘制连线的实现方法
/// </summary>
/// <param name="node1"></param>
/// <param name="node2"></param>
private void DrawPointToPointLine(FlowNode ctl1, FlowNode ctl2)
{
    // 定义两个点对象变量
    Point p1, p2;
    // 从左到右连线
    if (Math.Abs(ctl2.Location.X - ctl1.Location.X) > Math.Abs(ctl2.Location.Y -
        ctl1.Location.Y) && ctl2.Location.X >= ctl1.Location.X)
    {
        p1 = new Point(ctl1.Location.X + ctl1.Width + 1, ctl1.Location.Y + ctl1.
            Height / 2);
        p2 = new Point(ctl2.Location.X  - 1, ctl2.Location.Y + ctl2.Height / 2);
        DrawJoinLine(p1, p2, LineForward.L_R);
    }
    // 从右到左
    else if (Math.Abs(ctl2.Location.X - ctl1.Location.X) > Math.Abs(ctl2.Location.Y
        - ctl1.Location.Y) && ctl2.Location.X < ctl1.Location.X)
    {
        p1 = new Point(ctl1.Location.X - 1, ctl1.Location.Y + ctl1.Height / 2);
        p2 = new Point(ctl2.Location.X + ctl2.Width + 1, ctl2.Location.Y + ctl2.
            Height / 2);
```

```
22          DrawJoinLine(p1, p2, LineForward.R_L);
23      }
24      // 从上到下
25      else if (Math.Abs(ctl2.Location.X - ctl1.Location.X) <= Math.Abs(ctl2.Location.Y
            - ctl1.Location.Y) && ctl2.Location.Y >= ctl1.Location.Y)
26      {
27          p1 = new Point(ctl1.Location.X + ctl1.Width / 2, ctl1.Location.Y + ctl1.
                Height + 1);
28          p2 = new Point(ctl2.Location.X + ctl2.Width / 2, ctl2.Location.Y - 1);
29          DrawJoinLine(p1, p2, LineForward.U_D);
30      }
31      // 从下到上
32      else if(Math.Abs(ctl2.Location.X - ctl1.Location.X) < Math.Abs(ctl2.Location.Y -
            ctl1.Location.Y) && ctl2.Location.Y < ctl1.Location.Y)
33      {
34          p1 = new Point(ctl1.Location.X + ctl1.Width / 2, ctl1.Location.Y - 1);
35          p2 = new Point(ctl2.Location.X + ctl2.Width / 2, ctl2.Location.Y + ctl2.
                Height + 1);
36          DrawJoinLine(p1, p2, LineForward.D_U);
37      }
38  }
```

2.6.2 画直线

```
1   /// <summary>
2   /// 连接线的实现方式
3   /// </summary>
4   /// <param name="node1"></param>
5   /// <param name="node2"></param>
6   private void DrawPointToPointLine(FlowNode ctl1, FlowNode ctl2)
7   {
8       // 定义两个点对象变量
9       Point p1, p2;
10      // 从左到右连线
11      if (Math.Abs(ctl2.Location.X - ctl1.Location.X) > Math.Abs(ctl2.Location.Y -
            ctl1.Location.Y) && ctl2.Location.X >= ctl1.Location.X)
12      {
13          p1 = new Point(ctl1.Location.X + ctl1.Width + 1, ctl1.Location.Y + ctl1.
                Height / 2);
14          p2 = new Point(ctl2.Location.X - 1, ctl2.Location.Y + ctl2.Height / 2);
15          DrawJoinLine(p1, p2, LineForward.L_R);
16      }
17      // 从右到左
18      else if (Math.Abs(ctl2.Location.X - ctl1.Location.X) > Math.Abs(ctl2.Location.Y
            - ctl1.Location.Y) && ctl2.Location.X < ctl1.Location.X)
19      {
```

```
20          p1 = new Point(ctl1.Location.X - 1, ctl1.Location.Y + ctl1.Height / 2);
21          p2 = new Point(ctl2.Location.X + ctl2.Width + 1, ctl2.Location.Y + ctl2.
                Height / 2);
22          DrawJoinLine(p1, p2, LineForward.R_L);
23      }
24      // 从上到下
25      else if (Math.Abs(ctl2.Location.X - ctl1.Location.X) <= Math.Abs(ctl2.Location.Y
             - ctl1.Location.Y) && ctl2.Location.Y >= ctl1.Location.Y)
26      {
27          p1 = new Point(ctl1.Location.X + ctl1.Width / 2, ctl1.Location.Y + ctl1.
                Height + 1);
28          p2 = new Point(ctl2.Location.X + ctl2.Width / 2, ctl2.Location.Y - 1);
29          DrawJoinLine(p1, p2, LineForward.U_D);
30      }
31      // 从下到上
32      else if (Math.Abs(ctl2.Location.X - ctl1.Location.X) < Math.Abs(ctl2.Location.Y
            - ctl1.Location.Y) && ctl2.Location.Y < ctl1.Location.Y)
33      {
34          p1 = new Point(ctl1.Location.X + ctl1.Width / 2, ctl1.Location.Y - 1);
35          p2 = new Point(ctl2.Location.X + ctl2.Width / 2, ctl2.Location.Y + ctl2.
                Height + 1);
36          DrawJoinLine(p1, p2, LineForward.D_U);
37      }
38  }
39
40  /// <summary>
41  /// 绘制两点之间的连线
42  /// </summary>
43  /// <param name="p1"></param>
44  /// <param name="p2"></param>
45  /// <param name="l_R"></param>
46  private void DrawJoinLine(Point p1, Point p2, LineForward forward)
47  {
48      // 绘制图形的面板应该
49      Graphics g = panel1.CreateGraphics();
50      g.SmoothingMode = SmoothingMode.HighQuality;
51
52      // 绘制图形的线
53      Color color = Color.DarkRed;    // 创建一个color对象
54      Pen p = new Pen(color, 3);    // pen对象
55      p.DashStyle = DashStyle.Solid;    // 实线
56      p.StartCap = LineCap.Round;    // 起点端帽样式  圆形
57      p.EndCap = LineCap.ArrowAnchor;    // 终点端帽样式  箭头形状
58      p.LineJoin = LineJoin.Round; // 连接处样式 圆角
59
60      // 连线中间的两个点
```

```
61      Point inflectPoint1;
62      Point inflectPoint2;
63      if (forward == LineForward.L_R || forward == LineForward.R_L)
64      {
65          inflectPoint1 = new Point((p1.X + p2.X) / 2, p1.Y);
66          inflectPoint2 = new Point((p1.X + p2.X) / 2, p2.Y);
67      }
68      else
69      {
70          inflectPoint1 = new Point(p1.X, (p1.Y + p2.Y) / 2);
71          inflectPoint2 = new Point(p2.X, (p1.Y + p2.Y) / 2);
72      }
73
74      // 从点p1到中间两个点到p2
75      Point[] points = new Point[] { p1, inflectPoint1, inflectPoint2, p2 };
76      g.DrawLines(p, points);
77  }
```

2.6.3 连线刷新

连线刷新处理主要实现就是在流程编辑窗口的事件里面选择 paint，然后在需要重新绘制时根据之前连接的节点来重新绘制。

```
1   private void panel1_Paint(object sender, PaintEventArgs e)
2   {
3       var control = sender as Control;
4       DrawAllLines(control);
5   }
6
7   /// <summary>
8   ///
9   /// </summary>
10  /// <param name="control"></param>
11  private void DrawAllLines(Control control)
12  {
13      // 找出流程编辑器上所有流程控件
14      foreach (var ctl1 in control.Controls)
15      {
16          if (ctl1 is FlowNode)
17          {
18              var fn1 = (FlowNode)ctl1;
19              foreach (var ctl2 in control.Controls)
20              {
21                  if (ctl2 is FlowNode)
22                  {
23                      // 判断两个控件之间是否存在关系
```

```
24                      var fn2 = (FlowNode)ctl2;
25                      // 判断节点之间是否存在从属关系
26                      if (fn1.nextNodeID != null && fn1.nextNodeID == fn2.NodeID)
27                      {
28                          DrawPointToPointLine(fn1, fn2);
29                      }
30                  }
31              }
32          }
33      }
34  }
```

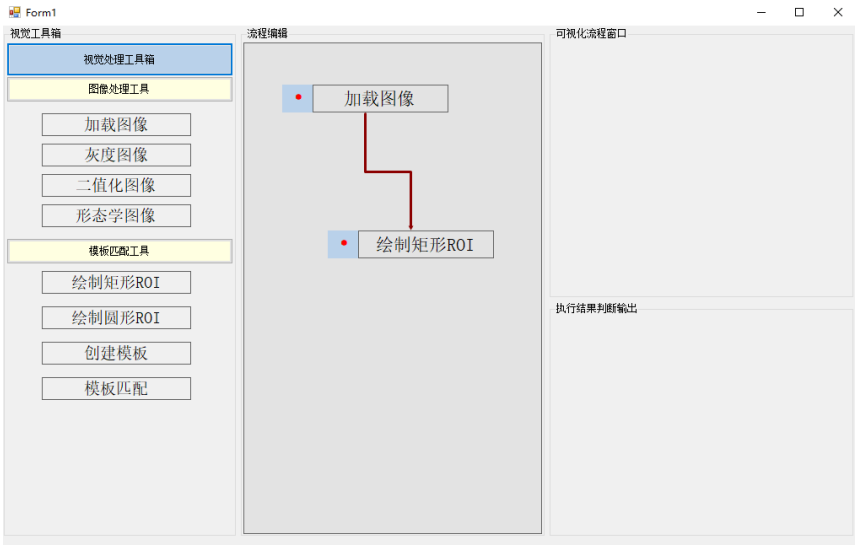

图 2.2: 连线

3 工具箱中的方法实现

3.1 流程节点 FlowNode 处理

这里的作用就是在流程编辑窗口中点击 FlowNode 能够使用其方法。

FlowNode 里面,小原点和文件部分都需要连接双击按钮事件,在属性的双击事件中选择”MouseDoubleClick”.

```
/// <summary>
/// 双击流程控件需要执行的逻辑
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void FlowNode_MouseDoubleClick(object sender, MouseEventArgs e)
{
    // 获取当前双击的流程节点的标题
    var title = this.NodeName;
    // 根据标题选择执行的流程逻辑
    switch (title)
    {
        case "加载图像":
        // 打开加载图像设置的对话框
        new LoadImageFrm(this).ShowDialog();
        previousNode = this;
            break;

        case "绘制矩形ROI":
        new DrawROIFrm(previousNode, this).ShowDialog();
            break;
        default:
            break;
        }
}
```

3.2 图像加载

先新建一个文件夹，用来存放工具箱中的 FlowNode 节点的方法。

然后在文件夹中添加窗体控件。

加载图像主要就两个按钮，一个加载，一个关闭。

```
using halconCSharp2.VControl;
using HalconDotNet;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
```

```
8  using System.Linq;
9  using System.Text;
10 using System.Threading.Tasks;
11 using System.Windows.Forms;
12
13 namespace halconCSharp2.VFrame
14 {
15     public partial class LoadImageFrm : Form
16     {
17         private FlowNode flowNode;
18
19         public LoadImageFrm(FlowNode flowNode)
20         {
21             InitializeComponent();
22             this.flowNode = flowNode;    // 方便后面设置节点
23         }
24
25         /// <summary>
26         /// 对象输入图像进行对应设置
27         /// </summary>
28         /// <param name="sender"></param>
29         /// <param name="e"></param>
30         private void btn_loadImage_Click(object sender, EventArgs e)
31         {
32             var dialog = new OpenFileDialog();
33             dialog.Filter = "选择图像类型␣|␣*.png;*.jpg;*.bmp";
34             if (dialog.ShowDialog() == DialogResult.OK)
35             {
36                 // 获取用户选择图像路径
37                 var imgUrl = dialog.FileName;
38                 // 根据获取图像路径创建Halcon图像对象
39                 var himg = new HImage(imgUrl);     // 图像
40                 System.Diagnostics.Debug.WriteLine(imgUrl);
41                 // 设置当前节点输出结果
42                 flowNode.output = himg;
43             }
44         }
45
46         private void btn_closewindow_Click(object sender, EventArgs e)
47         {
48             this.Close();
49         }
50     }
51 }
```

3.3 模板加载

3.3.1 halcon 的矩形控制新建

在 VControl 文件夹下添加” 用户控件 (windows 窗体)”,然后双击控件界面,进入”MyHalconControl_-Load” 函数体中，注意，这边实例化 HSmartWindowControl 对象，因为 HWindowControl 有一些版本之类的报错，很麻烦。然后设置一下显示图像的函数”DisplayImage”.

```
1  using HalconDotNet;
2  using System;
3  using System.Collections.Generic;
4  using System.ComponentModel;
5  using System.Data;
6  using System.Drawing;
7  using System.Linq;
8  using System.Text;
9  using System.Threading.Tasks;
10 using System.Windows.Forms;
11
12 namespace halconCSharp2.VControl
13 {
14     public partial class MyHalconControl : UserControl
15     {
16         private HWindowControl halconCtl;
17         public HWindow halconWindow { get; private set; }
18
19         public MyHalconControl()
20         {
21             InitializeComponent();
22         }
23
24         private void MyHalconControl_Load(object sender, EventArgs e)
25         {
26             //halconCtl = new HSmartWindowControl();
27             halconCtl = new HWindowControl();
28             halconWindow = halconCtl.HalconWindow;
29             // 设置布局
30             halconCtl.Dock = DockStyle.Fill;
31             // 添加到对应容器中
32             this.Controls.Add(halconCtl);
33         }
34         /// <summary>
35         /// 显示图像
36         /// </summary>
37         /// <param name="image"></param>
38         public void DisplayImage(HObject image)
39         {
```

```
40              ImgIsNotStretchDisplay(image, halconWindow);
41          }
42
43          /// <summary>
44          /// 实现图像等比缩放显示
45          /// </summary>
46          /// <param name="L_Img"></param>
47          /// <param name="Hwindow"></param>
48          public void ImgIsNotStretchDisplay(HObject L_Img, HTuple Hwindow)
49          {
50              HOperatorSet.ClearWindow(Hwindow);
51              HOperatorSet.GetImageSize(L_Img, out var width, out var height);
52              HOperatorSet.GetWindowExtents(Hwindow, out var _, out var _, out var
                    width2, out var height2);
53              HTuple hTuple = 1.0 * height2 / width2 * width;
54              if (hTuple > height)
55              {
56                  hTuple = 1.0 * (hTuple - height) / 2;
57                  HOperatorSet.SetPart(Hwindow, -hTuple, 0, hTuple + height, width);
58              }
59              else
60              {
61                  HTuple hTuple2 = 1.0 * width2 / height2 * height;
62                  hTuple2 = 1.0 * (hTuple2 - width) / 2;
63                  HOperatorSet.SetPart(Hwindow, 0, -hTuple2, height, hTuple2 + width);
64              }
65
66              HOperatorSet.DispObj(L_Img, Hwindow);
67          }
68      }
69  }
```

这里注意一下，第 27 行，自定义控件的时候使用的 HSmartWindowControl，照理说应该用 HWindowControl, 因为我使用的时候的版本问题，VS2019，Halcon 使用的是 17.12, 所以就是会有个问题，自定义好控件之后，将其拖到 ROI 界面的时候，如果用 x86 会有版本不匹配消息，用 x64 的话就是从工具箱拖到界面的时候控件直接从工具箱消失。

所以我这边的解决办法就是先用 HSmartWindowControl，然后等这个控件拖到自定义的 ROI 界面里面的时候，再回到自定义控件那里，将其改为 HWindowControl。

为什么要用 HWindowControl 呢，因为 HSmartWindowControl 加载图像之后，这个图像可以用鼠标拖动这些操作，以至于后面添加绿色框框选的时候会出错。而 HWindowControl 是静态的，也就是图片本身不会动，方便后面加框。

此问题卡了两天。

3.3.2 绘制矩形 ROI 界面设置

首先在设计器中设置好界面，如图3.1所示：

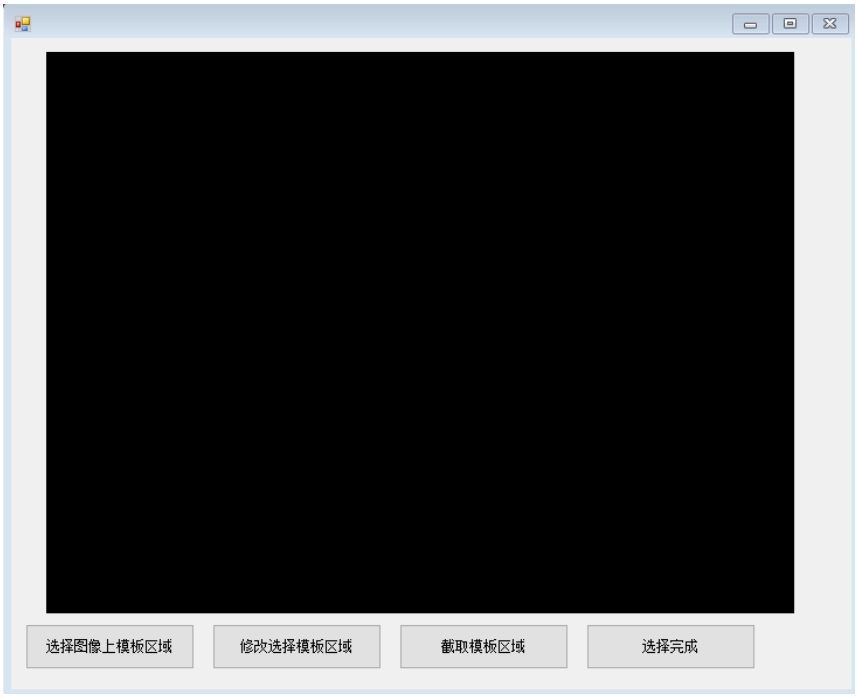

图 3.1: 绘制 ROI 界面

”选择图像上模板区域”这个按钮实现的功能，首先让其控件获得焦点，然后在使用”HWindow”这个类里面的”DrawRectangle1”功能进行绘制矩形框，这个矩形框是 Halcon 实现的功能，这个功能可以让其在界面上进行拖拽使用。

```
1         /// <summary>
2 /// 绘制矩形ROI
3 /// </summary>
4 /// <param name="sender"></param>
5 /// <param name="e"></param>
6 private void button1_Click(object sender, EventArgs e)
7 {
8     // 首先让其控件获取焦点
9     mhc.Focus();
10
11     // 绘制矩形ROI
12     hv_Handle.DrawRectangle1(out row1, out column1, out row2, out column2);  // 显示
          图像的控件
13     // 显示绘制矩形
14     hv_Handle.DispRectangle1(row1, column1, row2, column2);
15 }
```

修改选择模板区域这个功能，实现的思路首先清空窗口，然后在界面上显示图像。最后使用

DrawRectangle1Mod 函数在原来的界面上差不多原来的位置绘制矩形框，并将本次绘制的矩形框在界面上显示。

```
        /// <summary>
/// 修改ROI
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void button2_Click(object sender, EventArgs e)
{
    // 清空窗口显示数据
    hv_Handle.ClearWindow();
    // 重新显示图片
    mhc.DisplayImage(preNode.output);
    // 绘制交互式的矩形
    hv_Handle.DrawRectangle1Mod(row1, column1, row2, column2, out row1, out column1,
        out row2, out column2);
    // 显示绘制矩形
    hv_Handle.DispRectangle1(row1, column1, row2, column2);
}
```

截取模板区域按钮，就是截取刚才绘制的矩形框的区域在界面上进行绘制。

```
        /// <summary>
/// 截取模板区域并显示到对应视图窗口
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void button3_Click(object sender, EventArgs e)
{
    // 清空窗口显示数据
    hv_Handle.ClearWindow();
    // 根据绘制矩形坐标生成对应矩形对象
    HOperatorSet.GenRectangle1(out HObject rect, row1, column1, row2, column2);
    // 获取模板区域显示图像
    HOperatorSet.ReduceDomain(preNode.output, rect, out HObject ImageReduced);
    // 获取模板图像
    HOperatorSet.CropDomain(ImageReduced, out HObject ImagePart);
    // 重新显示图像
    mhc.DisplayImage(ImagePart);
    // 设置当前节点的输入与输出
    currentNode.input = preNode.output;
    currentNode.output = ImagePart;
    currentNode.RoiOutput = new HTuple[] { row1, column1, row2, column2 };
}
```

3.4 模板匹配

3.4.1 布局设置

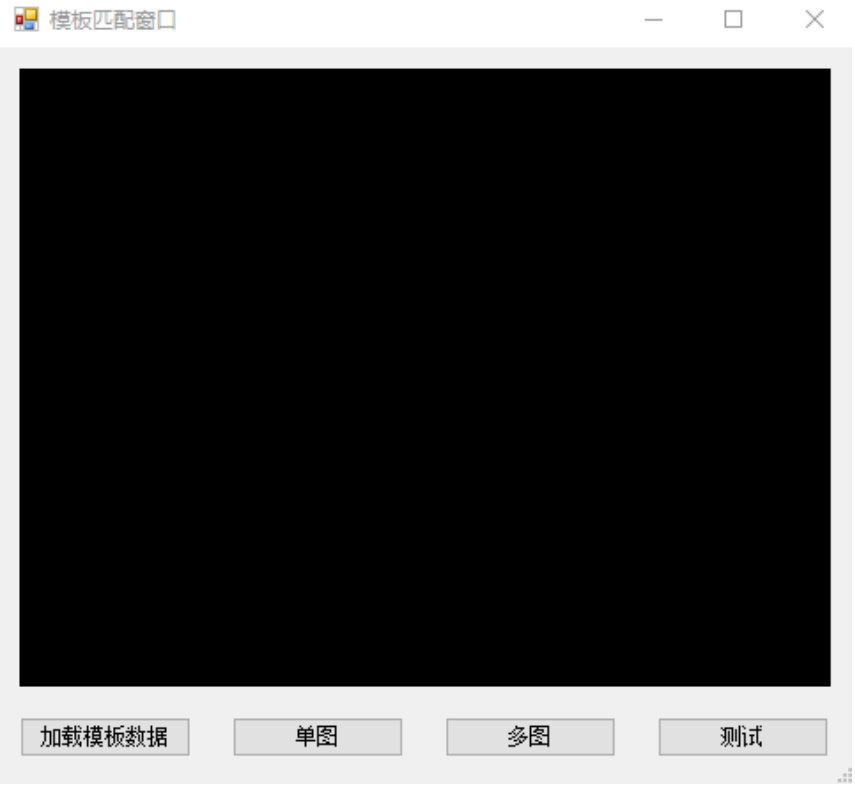

图 3.2: 在界面上绘制区域

加载模板数据按钮 c++

```
        /// <summary>
/// 窗体加载
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void button1_Click(object sender, EventArgs e)
{
    // 创建一个对话框
    var dialog = new OpenFileDialog();
    dialog.Filter = "模板及轮廓 | *.shm;*.hobj";
    dialog.RestoreDirectory = true;
    dialog.Multiselect = true;
    if(dialog.ShowDialog() == DialogResult.OK)
    {
        // 获取选择的轮廓和模板路径
        var urls = dialog.FileNames;
        // 使用linq语句
```

```
        var modelUrl = urls.Where(url => url.Contains(".shm")).FirstOrDefault();
        var xldUrl = urls.Where(url => url.Contains(".hobj")).FirstOrDefault();
        // 把轮廓及模板加载到内存中
        HOperatorSet.ReadShapeModel(modelUrl, out modelId);
        HOperatorSet.ReadObject(out modelXld, xldUrl);

    }
}
```

单独加载和多图加载

```
/// <summary>
/// 单图加载
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void button2_Click(object sender, EventArgs e)
{
    // 创建一个对话框
    var dialog = new OpenFileDialog();
    dialog.Filter = "图片|*.jpg;*.bmp;*.png";
    dialog.RestoreDirectory = true;
    if (dialog.ShowDialog() == DialogResult.OK)
    {
        var url = dialog.FileName;
        himage = new HImage(url);
        myHalconControl1.DisplayImage(himage);
        isMulti = false;
    }
}
/// <summary>
/// 多图加载
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void button3_Click(object sender, EventArgs e)
{
    // 创建一个对话框
    var dialog = new OpenFileDialog();
    dialog.Filter = "图片|*.jpg;*.bmp;*.png";
    dialog.RestoreDirectory = true;
    dialog.Multiselect = true;
    if (dialog.ShowDialog() == DialogResult.OK)
    {
        var urls = dialog.FileNames;
        foreach (var url in urls)
        {
```

```
            var hImg = new HImage(url);
            // 存储到图像集合中
            hImgs.Add(hImg);
        }
        myHalconControl1.DisplayImage(hImgs[0]);
        isMulti = true;
    }
}
```

模板匹配按钮

```
/// <summary>
/// 测试创建的模板是否匹配选择的测试图像
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void button4_Click(object sender, EventArgs e)
{
    if(!isMulti)
    {
        // 单图匹配
        ModelMatch(himage);
    }
    else
    {
        // 多图匹配
        timer.Start();
        timer.Interval = 1000;
        timer.Tick += Timer_Tick;    // 绑定定时器事件
    }
}

int index = 0;
/// <summary>
/// 使用定时器来轮询校验多张图像匹配
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void Timer_Tick(object sender, EventArgs e)
{
    if (hImgs.Count > 0)
    {
        // 获取一张图像
        var img = hImgs[index++];
        // 执行图像匹配
        ModelMatch(img);
        // 判断模板匹配是否完成
```

```
37          if (index == hImgs.Count)
38          {
39              timer.Stop();
40              index = 0;
41              hv_Handle.ClearWindow();
42                  }
43          }
44  }
45
46  private void ModelMatch(HObject L_Image)
47  {
48      HTuple hv_Score = new HTuple();
49      // 通过模板ID定位模板图像位置及角度
50      // L_Image 输入图像
51      // modelID 形状模型ID
52      // 0(最小角度)
53      // (new HTuple(360)).TupleRad() 最大角度
54      // 0.5 最小缩小比例
55      // 1   最大缩放比例
56      // 0.5 最小分数
57      // least_square 优化方法
58      // 0   金字塔等级
59      // 0.7 最大角度误差
60      // hv_Row1  输出的行坐标
61      // hv_Column1 输出的列坐标
62      // hv_Angle  输出的角度
63      // hv_Score  输出的匹配得分
64      HOperatorSet.FindShapeModel(L_Image, modelId, 0, (new HTuple(360)).TupleRad()
65      , 0.5, 1, 0.5, "least_squares", 0, 0.7, out HTuple hv_Row1, out HTuple
            hv_Column1,
66      out HTuple hv_Angle, out hv_Score);
67      if (hv_Score.Type == HTupleType.EMPTY)
68      {
69          HOperatorSet.DispText(hv_Handle, "配置失败",
70          "window", 12, 12, "red", new HTuple(), new HTuple());
71          return;
72      }
73      // 这边的主要作用就是通过模板匹配得到的匹配结果进行旋转和平移操作,
74      // 然后将这个变换应用到模板轮廓上, 以便在图像中正确地显示出匹配到的轮廓
75
76      // 创建一个齐次矩阵, 创建一个2D单位矩阵
77      HOperatorSet.HomMat2dIdentity(out HTuple hv_HomMat2DIdentity);
78      //模板仿射变换, hv_Row1 Y方向平移量, hv_Column1 X方向平移量, 输出2D仿射变换矩阵
79      HOperatorSet.HomMat2dTranslate(hv_HomMat2DIdentity, hv_Row1, hv_Column1,
80      out HTuple hv_HomMat2DTranslate);
81      // 旋转
```

```
82      HOperatorSet.HomMat2dRotate(hv_HomMat2DTranslate, hv_Angle, hv_Row1, hv_Column1,
83      out HTuple hv_HomMat2DRotate);
84      // 仿射变换轮廓
85      HOperatorSet.AffineTransContourXld(modelXld, out HObject ho_ContoursAffineTrans,
86      hv_HomMat2DRotate);
87      // 显示结果
88      HOperatorSet.SetColor(hv_Handle, "red");
89      HOperatorSet.SetLineWidth(hv_Handle, 2);
90      HOperatorSet.DispObj(L_Image, hv_Handle);
91      HOperatorSet.DispObj(ho_ContoursAffineTrans, hv_Handle);
92      // 获取轮廓中心
93      HOperatorSet.AreaCenterXld(ho_ContoursAffineTrans, out HTuple Area, out HTuple
            xldRow, out HTuple xldColumn, out HTuple _);
94      HOperatorSet.SetColor(hv_Handle, "green");
95      HOperatorSet.SetLineWidth(hv_Handle, 1);
96      // 绘制十字准星
97      HOperatorSet.DispCross(hv_Handle, xldRow, xldColumn, 60, 0);
98      HOperatorSet.DispText(hv_Handle, "匹配坐标: (row=" + hv_Row1.TupleString(".2f")
            + ",column=" + hv_Column1.TupleString(".2f"),
99      "window", 12, 12, "red", new HTuple(), new HTuple());
100
101     // 给父类的静态变量传值
102     Form1.centerX = hv_Row1;
103     Form1.centerY = hv_Column1;
104 }
```

4　整体流程单步执行和单次执行

最后设计的界面如图4.1所示，流程编辑下面有单步执行和单次执行，可视化流程窗口显示每次执行的图片显示，执行现状以及结果判断输出。

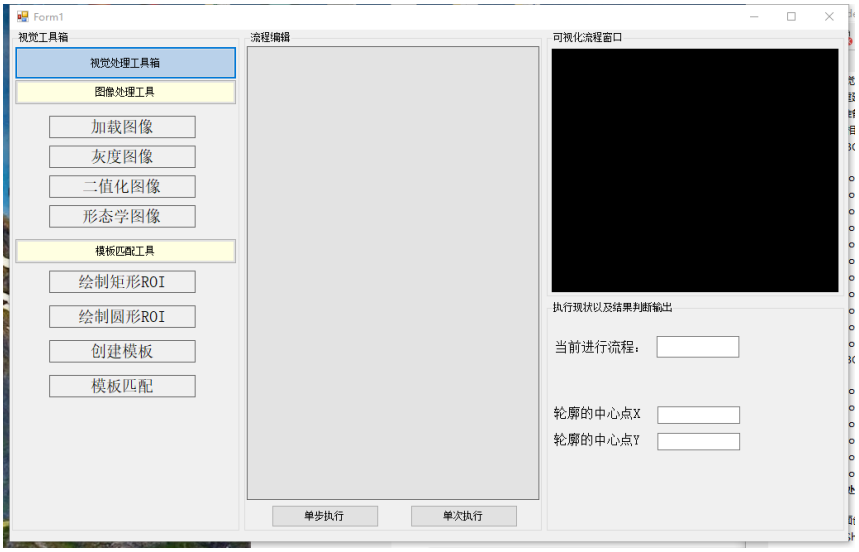

图 4.1: 整体界面

单次执行的实现步骤如下，主要思路是遍历所有的节点，然后从节点中找到没有上节点的，说明这个节点是最开始的执行步骤，然后依次找下一个节点，将其放入数组中，以便下一个节点调用，然后每次点击按钮的时候，有个 ClickNum 自动加 1：然后就判断每个节点表示什么内容，然后进行显示一下。

模板匹配环节，使用的函数也是先调整图片尺寸适合界面，然后使用 ModelMatch 功能进行匹配检测。

```
1  static int ClickNum = 0;
2  /// <summary>
3  /// 单步执行
4  /// </summary>
5  /// <param name="sender"></param>
6  /// <param name="e"></param>
7  private void button2_Click(object sender, EventArgs e)
8  {
9      // 第一次点击的时候
10     if (ClickNum == 0)
11     {
12         System.Diagnostics.Debug.WriteLine(nodeList.Count());
13         foreach (var nodelist in nodeList)
14         {
15             if (nodelist.preNodeID == null)
16             {
```

```
17                  beginNode = nodelist;
18              }
19              if(nodelist.NodeName == "加载图像")
20              {
21                  RawImage = nodelist.output;
22                  myWinForm.DisplayImage(RawImage);
23              }
24          }
25
26          orderNodeList.Add(beginNode);
27          var currentNode = beginNode;
28          while (currentNode.nextNodeID != null)
29          {
30              foreach (var node in nodeList)
31              {
32                  if(node.NodeID == currentNode.nextNodeID)
33                  {
34                      orderNodeList.Add(node);
35                      currentNode = node;
36                  }
37              }
38          }
39
40          beginNode.LightColor = Color.Blue;
41          ClickNum++;
42      }
43      else
44      {
45          if (ClickNum == orderNodeList.Count)
46          {
47              ClickNum = 0;
48              foreach (var node in orderNodeList)
49              {
50                  node.LightColor = Color.Red;
51              }
52              currentStep.Text = orderNodeList[ClickNum].NodeName;
53              myWinForm.DisplayImage(RawImage);
54          }
55
56          orderNodeList[ClickNum].LightColor = Color.Blue;                    // 流
                程编辑上面颜色置为蓝色
57          System.Diagnostics.Debug.WriteLine(orderNodeList[ClickNum].NodeName);  // 打
                印输出
58
59          if (orderNodeList[ClickNum].NodeName == "绘制矩形ROI")
60          {
```

```
61              currentStep.Text = "绘制矩形ROI";
62          }
63          else if(orderNodeList[ClickNum].NodeName == "创建模板")
64          {
65              currentStep.Text = "创建模板";
66
67              // 当前节点的输出参数图像
68              HObject image = orderNodeList[ClickNum].output;
69              myWinForm.DisplayImage(image);
70          }
71          else if(orderNodeList[ClickNum].NodeName == "模板匹配")
72          {
73              currentStep.Text = "模板匹配";
74
75              adjustImage(RawImage);     // 调整尺寸
76              ModelMatch(RawImage);      // 进行匹配
77
78              // 显示模板center位置
79              CenterX.Text = centerX.ToString();     // 将检测到的结果输出到指定区域
80              CenterY.Text = centerY.ToString();     // 将检测到的结果输出到指定区域
81          }
82          ClickNum++;
83      }
84  }
```

adjustImage 函数实现

```
1   public void adjustImage(HObject image)
2   {
3       HOperatorSet.ClearWindow(myWinForm.halconWindow);
4       HOperatorSet.GetImageSize(image, out var width, out var height);
5       HOperatorSet.GetWindowExtents(myWinForm.halconWindow, out var _, out var _, out
            var width2, out var height2);
6       HTuple hTuple = 1.0 * height2 / width2 * width;
7       if (hTuple > height)
8       {
9           hTuple = 1.0 * (hTuple - height) / 2;
10          HOperatorSet.SetPart(myWinForm.halconWindow, -hTuple, 0, hTuple + height,
                width);
11      }
12      else
13      {
14          HTuple hTuple2 = 1.0 * width2 / height2 * height;
15          hTuple2 = 1.0 * (hTuple2 - width) / 2;
16          HOperatorSet.SetPart(myWinForm.halconWindow, 0, -hTuple2, height, hTuple2 +
                width);
17          System.Diagnostics.Debug.WriteLine($"SetPart:␣(0,␣-{hTuple2},␣{height},␣{
```

```
            hTuple2 + width})");
    }

    HOperatorSet.DispObj(image, myWinForm.halconWindow);
}
```

ModelMatch 函数实现

```
private void ModelMatch(HObject img)
{
    HTuple hv_Score = new HTuple();
    // 通过模板ID定位模板图像位置及角度
    // L_Image 输入图像
    // modelID 形状模型ID
    // 0(最小角度)
    // (new HTuple(360)).TupleRad() 最大角度
    // 0.5 最小缩小比例
    // 1   最大缩放比例
    // 0.5 最小分数
    // least_square 优化方法
    // 0   金字塔等级
    // 0.7 最大角度误差
    // hv_Row1  输出的行坐标
    // hv_Column1 输出的列坐标
    // hv_Angle  输出的角度
    // hv_Score  输出的匹配得分
    HOperatorSet.FindShapeModel(img, hv_ModelID, 0, (new HTuple(360)).TupleRad()
    , 0.5, 1, 0.5, "least_squares", 0, 0.7, out HTuple hv_Row1, out HTuple
        hv_Column1,
    out HTuple hv_Angle, out hv_Score);

    if (hv_Score.Type == HTupleType.EMPTY)
    {
        HOperatorSet.DispText(myWinForm.halconWindow, "配置失败",
        "window", 12, 12, "red", new HTuple(), new HTuple());
        return;
    }
    // 创建一个齐次矩阵, 创建一个2D单位矩阵
    HOperatorSet.HomMat2dIdentity(out HTuple hv_HomMat2DIdentity);
    //模板仿射变换, hv_Row1 Y方向平移量, hv_Column1 X方向平移量, 输出2D仿射变换矩阵
    HOperatorSet.HomMat2dTranslate(hv_HomMat2DIdentity, hv_Row1, hv_Column1,
    out HTuple hv_HomMat2DTranslate);
    // 旋转
    HOperatorSet.HomMat2dRotate(hv_HomMat2DTranslate, hv_Angle, hv_Row1, hv_Column1,
    out HTuple hv_HomMat2DRotate);
    // 仿射变换轮廓
    HOperatorSet.AffineTransContourXld(modelContours, out HObject
```

```
        ho_ContoursAffineTrans,
39      hv_HomMat2DRotate);
40      // 显示结果
41      HOperatorSet.SetColor(myWinForm.halconWindow, "red");
42      HOperatorSet.SetLineWidth(myWinForm.halconWindow, 2);
43      HOperatorSet.DispObj(img, myWinForm.halconWindow);
44      HOperatorSet.DispObj(ho_ContoursAffineTrans, myWinForm.halconWindow);
45      // 获取轮廓中心
46      HOperatorSet.AreaCenterXld(ho_ContoursAffineTrans, out HTuple Area, out HTuple
            xldRow, out HTuple xldColumn, out HTuple _);
47      HOperatorSet.SetColor(myWinForm.halconWindow, "green");
48      HOperatorSet.SetLineWidth(myWinForm.halconWindow, 1);
49      // 绘制十字准星
50      HOperatorSet.DispCross(myWinForm.halconWindow, xldRow, xldColumn, 60, 0);
51      HOperatorSet.DispText(myWinForm.halconWindow, "匹配坐标: (row=" + hv_Row1.
            TupleString(".2f") + ",column=" + hv_Column1.TupleString(".2f"),
52      "window", 12, 12, "red", new HTuple(), new HTuple());
53
54      centerX = hv_Row1;
55      centerY = hv_Column1;
56
57  }
```

5 C# 程序导出

https://blog.csdn.net/chengcao123/article/details/126936543

https://blog.csdn.net/code_6o1/article/details/140298908

6 总结

此项目学习到的东西

(1) 按钮点击收起展开面板操作

这个的思路主要就是按钮按下的时候下面的 panel 进行隐藏和展开作用。

(2) 拖拉拽的实现思路

主要就是用 label 控件，当启动拖动的时候，使用 MouseDown 信号捕获这个事件，然后将其拖动过程中的时候，主要就是拖动过程中的视觉反馈更新，最重要的就是当拖动释放的时候，捕获当前鼠标的位置，然后在当前位置实现目标的放置。

另外就是在流程界面的时候，这个控件可以设置上下左右的边框位置。

(3) C# 中的右击弹窗事件

这个是拖动控件"ContextMenuStrip" 实现的,需要注意的是设置弹窗的控件右边"ContextMenuStrip" 这个选项需要选择"ContextMenuStrip" 名称。

(4) C# 中的版本对应

就是你电脑本地安装的什么版本的 halcon，你 C# 里面使用 nuget 就要使用什么版本的 halcon-dotnet，另外就是自定义控件的时候，HWindowControl 和 HSmartWindowControl 搞了很长时间。

(5) static 变量的使用

C# 中在类中创建了一个 static 变量，在别的文件的类中，可以直接使用`类.static变量`的方式来给这个变量赋值。

消息的传递本来打算是使用事件处理的，发现 C# 下使用静态变量能够直接传参。

目前这个层次能想到的改进的地方

(1) 像拖拉拽控件那边的实现部分，目前其实现类似链表，有前节点和后节点，但是其实也有个问题，就是如果多个输入情况或者多个输出的情况，这种方式就不能很好的实现。

(2) 像 C# 中的多个界面传参的方式，由于 C# 写界面不是很熟，不知道有没有事件机制，如果是 QT 框架下，各个界面之间使用信号槽机制传参感觉是挺方便的。

此项目来源：https://www.bilibili.com/video/BV1e24zeFEt7/