天筹-浑天可视化集成设计方案

一、项目背景

1.1 设计目标

基于现有的天筹优化决策系统和浑天验证仿真系统，实现以下功能整合：

1. **将可视化功能迁移到浑天页面**：将 `tianchou-integration-design.md` 中的布局可视化（轻工业）和 AGV 路径可视化（重工业）功能从天筹页面迁移到浑天仿真页面
2. **实现页面跳转**：在天筹优化页面添加跳转按钮，用户可以点击后跳转到浑天仿真页面查看可视化效果
3. **重点实现 AGV 路径可视化**：在浑天页面实现重工业场景下的 AGV 路径优化可视化

1.2 架构调整

原架构：

- 天筹页面：优化算法 + 方案展示 + 可视化
- 浑天页面：数字孪生仿真 + 压力测试

新架构：

- 天筹页面：优化算法 + 方案展示 + **跳转按钮**
- 浑天页面：数字孪生仿真 + 压力测试 + **布局/路径可视化**

二、功能设计

2.1 天筹页面改造

2.1.1 添加"查看仿真"按钮

在天筹页面的方案展示区域添加按钮，允许用户跳转到浑天页面查看可视化效果。

位置：

- 方案卡片（PlanCard）底部
- 页面底部操作栏

交互逻辑：

```
// 点击按钮时传递优化结果数据
const handleViewSimulation = (solution: ParetoSolution) => {
  navigate('/app/simulation', {
    state: {
      optimizationResult: {
        type: task.industry_type, // 'light' 或 'heavy'
        solution: solution,
        taskId: task.id
      }
    }
  })
}
```

2.1.2 数据传递结构

```
interface OptimizationResult {
  type: 'light' | 'heavy'  // 行业类型
  solution: ParetoSolution  // 选中的方案
  taskId: string            // 任务ID

  // 轻工业数据
  layoutData?: {
    workshopDimensions: { length: number; width: number }
    devices: Array<{
      id: number
      name: string
      originalPosition: [number, number]
      newPosition: [number, number]
      size: { width: number; height: number }
    }>
    movedDevices: Array<{
      deviceId: number
      distance: number
      cost: number
    }>
  }

  // 重工业数据
  agvData?: {
    stations: Array<{
      id: number
      name: string
```

```
      position: [number, number]
    }>
    agvRoutes: Array<{
      agvId: number
      route: Array<[number, number]>
      completionTime: number
      tasks: Array<{
        from: number
        to: number
        startTime: number
        endTime: number
      }>
    }>
    metrics: {
      totalCompletionTime: number
      bottleneckUtilization: number
    }
  }
}
```

2.2 浑天页面改造

2.2.1 新增可视化模式

在现有的浑天页面基础上，新增两种可视化模式：

```
type SimulationMode =
  | 'device_rearrangement'   // 设备重排（轻工业）
  | 'route_optimization'     // 路径优化（重工业）
  | 'stress_test'            // 压力测试（原有功能）
```

2.2.2 模式切换 UI

在页面顶部工具栏添加模式选择器：

```
<div className="flex items-center gap-4">
  <span className="text-xs text-slate-400 font-bold uppercase">
    可视化模式
  </span>
  <div className="flex bg-white/5 border border-white/10 rounded-xl p-1">
    <button
```

```
        onClick={() => setSimulationMode('device_rearrangement')}
        className={`px-3 py-1 text-xs font-bold rounded-lg ${
          simulationMode === 'device_rearrangement'
            ? 'bg-blue-600 text-white'
            : 'text-slate-400'
        }`}
      >
        设备布局优化
      </button>
      <button
        onClick={() => setSimulationMode('route_optimization')}
        className={`px-3 py-1 text-xs font-bold rounded-lg ${
          simulationMode === 'route_optimization'
            ? 'bg-blue-600 text-white'
            : 'text-slate-400'
        }`}
      >
        AGV路径优化
      </button>
      <button
        onClick={() => setSimulationMode('stress_test')}
        className={`px-3 py-1 text-xs font-bold rounded-lg ${
          simulationMode === 'stress_test'
            ? 'bg-blue-600 text-white'
            : 'text-slate-400'
        }`}
      >
        压力测试
      </button>
    </div>
  </div>
```

三、AGV 路径可视化详细设计

3.1 可视化需求

目标：在浑天页面的数字孪生画布上，动态展示 AGV 调度路径优化的过程和结果。

核心元素：

1. 工位节点（Stations）
2. AGV 小车（移动的图标）

3. 路径线（带方向的曲线）
4. 时间轴（显示调度进度）
5. 性能指标（完工时间、瓶颈利用率）

3.2 技术实现方案

3.2.1 使用 SVG + Framer Motion

优势：

- 轻量级，适合 Web 演示
- 支持平滑动画
- 易于交互

核心组件：

```
// AGVPathVisualizer.tsx
import { motion } from 'framer-motion'
import { useState, useEffect } from 'react'

interface Station {
  id: number
  name: string
  position: [number, number]
}

interface AGVRoute {
  agvId: number
  route: Array<[number, number]>
  completionTime: number
  color: string
}

const AGVPathVisualizer: React.FC<{
  stations: Station[]
  routes: AGVRoute[]
  isPlaying: boolean
  speed: number
}> = ({ stations, routes, isPlaying, speed }) => {
  const [progress, setProgress] = useState(0)

  // SVG 画布尺寸
  const canvasWidth = 1200
  const canvasHeight = 800
```

```
return (
  <svg
    width="100%"
    height="100%"
    viewBox={`0 0 ${canvasWidth} ${canvasHeight}`}
    className="absolute inset-0"
  >
    {/* 背景网格 */}
    <defs>
      <pattern
        id="grid"
        width="60"
        height="60"
        patternUnits="userSpaceOnUse"
      >
        <path
          d="M 60 0 L 0 0 0 60"
          fill="none"
          stroke="rgba(59, 130, 246, 0.1)"
          strokeWidth="1"
        />
      </pattern>
    </defs>
    <rect width="100%" height="100%" fill="url(#grid)" />

    {/* 工位节点 */}
    {stations.map(station => (
      <g key={station.id}>
        <circle
          cx={station.position[0]}
          cy={station.position[1]}
          r={30}
          fill="rgba(59, 130, 246, 0.2)"
          stroke="#3b82f6"
          strokeWidth={2}
        />
        <text
          x={station.position[0]}
          y={station.position[1]}
          textAnchor="middle"
          dy=".3em"
          fill="white"
```

```
            fontSize={12}
            fontWeight="bold"
          >
            {station.name}
          </text>
        </g>
      ))}

      {/* AGV 路径线 */}
      {routes.map(route => (
        <motion.path
          key={route.agvId}
          d={generatePathD(route.route)}
          stroke={route.color}
          strokeWidth={3}
          fill="none"
          strokeDasharray="10,5"
          initial={{ pathLength: 0 }}
          animate={{ pathLength: isPlaying ? 1 : 0 }}
          transition={{
            duration: route.completionTime / speed,
            ease: "linear"
          }}
        />
      ))}

      {/* AGV 小车 */}
      {routes.map(route => (
        <AGVIcon
          key={`agv-${route.agvId}`}
          route={route.route}
          color={route.color}
          isPlaying={isPlaying}
          speed={speed}
          completionTime={route.completionTime}
        />
      ))}
    </svg>
  )
}

// 生成 SVG 路径字符串
const generatePathD = (points: Array<[number, number]>): string => {
```

```
  if (points.length < 2) return ''

  let path = `M ${points[0][0]} ${points[0][1]}`

  for (let i = 1; i < points.length; i++) {
    const prev = points[i - 1]
    const curr = points[i]
    const next = points[i + 1]

    if (next) {
      // 使用贝塞尔曲线平滑路径
      const cp1x = prev[0] + (curr[0] - prev[0]) * 0.5
      const cp1y = prev[1] + (curr[1] - prev[1]) * 0.5
      path += ` Q ${cp1x} ${cp1y} ${curr[0]} ${curr[1]}`
    } else {
      path += ` L ${curr[0]} ${curr[1]}`
    }
  }

  return path
}

// AGV 图标组件
const AGVIcon: React.FC<{
  route: Array<[number, number]>
  color: string
  isPlaying: boolean
  speed: number
  completionTime: number
}> = ({ route, color, isPlaying, speed, completionTime }) => {
  return (
    <motion.g
      initial={{ offsetDistance: "0%" }}
      animate={{
        offsetDistance: isPlaying ? "100%" : "0%"
      }}
      transition={{
        duration: completionTime / speed,
        ease: "linear",
        repeat: isPlaying ? Infinity : 0
      }}
    >
      <motion.circle
```

```
          cx={route[0][0]}
          cy={route[0][1]}
          r={8}
          fill={color}
          className="drop-shadow-lg"
        />
        <motion.path
          d={`M ${route[0][0] - 5} ${route[0][1] - 3}
              L ${route[0][0] + 5} ${route[0][1]}
              L ${route[0][0] - 5} ${route[0][1] + 3} Z`}
          fill="white"
        />
      </motion.g>
    )
}

export default AGVPathVisualizer
```

3.3 数据流设计

3.3.1 从天筹接收数据

```
// Huntian.tsx
const Huntian: React.FC = () => {
  const location = useLocation()
  const [optimizationData, setOptimizationData] =
useState<OptimizationResult | null>(null)

  useEffect(() => {
    if (location.state?.optimizationResult) {
      const data = location.state.optimizationResult as OptimizationResult
      setOptimizationData(data)

      // 根据类型设置模式
      if (data.type === 'heavy') {
        setSimulationMode('route_optimization')
      } else if (data.type === 'light') {
        setSimulationMode('device_rearrangement')
      }
    }
  }, [location.state])

  return (
```

```
    // ...
  )
}
```

3.3.2 Mock 数据示例

```
const mockAGVData = {
  stations: [
    { id: 1, name: '上料区', position: [200, 200] },
    { id: 2, name: '加工区A', position: [500, 200] },
    { id: 3, name: '加工区B', position: [800, 200] },
    { id: 4, name: '检测区', position: [500, 500] },
    { id: 5, name: '下料区', position: [200, 500] },
  ],
  agvRoutes: [
    {
      agvId: 1,
      route: [
        [200, 200], // 上料区
        [500, 200], // 加工区A
        [500, 500], // 检测区
        [200, 500], // 下料区
      ],
      completionTime: 120, // 秒
      color: '#3b82f6',
      tasks: [
        { from: 1, to: 2, startTime: 0, endTime: 30 },
        { from: 2, to: 4, startTime: 30, endTime: 80 },
        { from: 4, to: 5, startTime: 80, endTime: 120 },
      ]
    },
    {
      agvId: 2,
      route: [
        [200, 200],
        [800, 200],
        [500, 500],
        [200, 500],
      ],
      completionTime: 150,
      color: '#10b981',
      tasks: [
        { from: 1, to: 3, startTime: 0, endTime: 50 },
```

```
        { from: 3, to: 4, startTime: 50, endTime: 110 },
        { from: 4, to: 5, startTime: 110, endTime: 150 },
      ]
    }
  ],
  metrics: {
    totalCompletionTime: 150,
    bottleneckUtilization: 0.85
  }
}
```

四、布局可视化设计（轻工业）

4.1 可视化需求

展示车间设备布局优化前后的对比，包括：

1. 设备原始位置
2. 设备优化后位置
3. 移动路径和距离
4. 搬运频率热力图

4.2 实现方案

```
// LayoutVisualizer.tsx
const LayoutVisualizer: React.FC<{
  layoutData: LayoutData
  isPlaying: boolean
}> = ({ layoutData, isPlaying }) => {
  const [showOptimized, setShowOptimized] = useState(false)

  return (
    <div className="relative w-full h-full">
      {/* 车间边界 */}
      <div
        className="absolute border-2 border-blue-500/30"
        style={{
          width: `${layoutData.workshopDimensions.length}px`,
          height: `${layoutData.workshopDimensions.width}px`
        }}
      >
```

```
        {/* 设备 */}
        {layoutData.devices.map(device => (
          <motion.div
            key={device.id}
            className="absolute bg-blue-600 rounded-lg flex items-center
justify-center text-white text-xs font-bold"
            style={{
              width: device.size.width,
              height: device.size.height
            }}
            initial={{
              x: device.originalPosition[0],
              y: device.originalPosition[1]
            }}
            animate={{
              x: showOptimized ? device.newPosition[0] :
device.originalPosition[0],
              y: showOptimized ? device.newPosition[1] :
device.originalPosition[1]
            }}
            transition={{ duration: 2, ease: 'easeInOut' }}
          >
            {device.name}
          </motion.div>
        ))}

        {/* 移动路径指示 */}
        {showOptimized && layoutData.movedDevices.map(moved => {
          const device = layoutData.devices.find(d => d.id ===
moved.deviceId)
          if (!device) return null

          return (
            <svg
              key={`path-${moved.deviceId}`}
              className="absolute inset-0 pointer-events-none"
            >
              <motion.line
                x1={device.originalPosition[0]}
                y1={device.originalPosition[1]}
                x2={device.newPosition[0]}
                y2={device.newPosition[1]}
                stroke="#f59e0b"
```

```
                strokeWidth={2}
                strokeDasharray="5,5"
                initial={{ pathLength: 0 }}
                animate={{ pathLength: 1 }}
                transition={{ duration: 1.5 }}
              />
            </svg>
          )
        })}
      </div>

      {/* 控制按钮 */}
      <button
        onClick={() => setShowOptimized(!showOptimized)}
        className="absolute bottom-4 right-4 px-4 py-2 bg-blue-600 text-
white rounded-lg"
      >
        {showOptimized ? '显示原始布局' : '显示优化布局'}
      </button>
    </div>
  )
}
```

五、UI/UX 设计

5.1 页面布局

浑天页面新布局

```
┌──────────────────────────────────────────────────────────────┐
│  顶部工具栏                                                  │
│  [模式选择] [倍速] [播放/暂停] [重置]                        │
├──────────────────────────────────────────────────────────────┤
│                                                              │
│                      主可视化画布                            │
│                                                              │
│  ┌──────────────────────────────────────────────────┐        │
│  │                                                  │        │
│  │   [设备布局 / AGV路径 / 压力测试]                │        │
│  │                                                  │        │
│  │   - 设备布局模式: 显示设备移动动画               │        │
```

```
│   │    - AGV路径模式：显示AGV调度路径                    │      │
│   │    - 压力测试模式：显示故障注入测试                    │      │
│   │                                                    │      │
│   └────────────────────────────────────────────────────┘      │
│                                                               │
├───────────────────────────────────────────────────────────────┤
│  底部信息栏                                                    │
│  [进度条] [性能指标] [日志输出]                                 │
└───────────────────────────────────────────────────────────────┘
```

5.2 交互流程

```
用户操作流程：
1. 在天筹页面查看优化方案
2. 点击"查看仿真"按钮
3. 跳转到浑天页面，自动加载对应模式
4. 点击"播放"按钮，观看可视化动画
5. 调整倍速，快速查看结果
6. 查看性能指标和ROI报告
7. 点击"推送执行"部署到物理层
```

5.3 视觉设计

配色方案：

- 主色调：深蓝色背景 (#050810)
- 强调色：天蓝色 (#3b82f6)
- 成功色：翠绿色 (#10b981)
- 警告色：琥珀色 (#f59e0b)
- 危险色：红色 (#ef4444)

动画效果：

- 路径绘制：使用 `pathLength` 动画
- 设备移动：使用 `easeInOut` 缓动
- AGV 移动：使用 `linear` 匀速
- 进度条：使用渐变色 + 发光效果

六、实施计划

Phase 1: 天筹页面改造 (1天)

任务：

- [ ] 在方案卡片添加"查看仿真"按钮
- [ ] 实现数据传递逻辑
- [ ] 测试路由跳转

文件修改：

- `frontend/pages/Tianchou.tsx`

Phase 2: 浑天页面基础改造 (1天)

任务：

- [ ] 添加模式切换器
- [ ] 接收天筹传递的数据
- [ ] 实现模式切换逻辑

文件修改：

- `frontend/pages/Huntian.tsx`

Phase 3: AGV 路径可视化实现 (2-3天)

任务：

- [ ] 创建 AGVPathVisualizer 组件
- [ ] 实现工位节点渲染
- [ ] 实现路径线绘制
- [ ] 实现 AGV 图标动画
- [ ] 添加时间轴控制
- [ ] 集成到浑天页面

新增文件：

- `frontend/components/AGVPathVisualizer.tsx`
- `frontend/components/StationNode.tsx`
- `frontend/components/AGVIcon.tsx`

Phase 4: 布局可视化实现 (2天)

任务：

- [ ] 创建 LayoutVisualizer 组件
- [ ] 实现设备渲染
- [ ] 实现移动动画

- [ ] 添加对比切换
- [ ] 集成到浑天页面

新增文件：

- `frontend/components/LayoutVisualizer.tsx`

Phase 5: 测试和优化 (1天)

任务：

- [ ] 端到端测试
- [ ] 性能优化
- [ ] 动画流畅度调优
- [ ] 响应式适配

七、技术要点

7.1 动画性能优化

```
// 使用 requestAnimationFrame 优化动画
const useAnimationFrame = (callback: (deltaTime: number) => void) => {
  const requestRef = useRef<number>()
  const previousTimeRef = useRef<number>()

  useEffect(() => {
    const animate = (time: number) => {
      if (previousTimeRef.current !== undefined) {
        const deltaTime = time - previousTimeRef.current
        callback(deltaTime)
      }
      previousTimeRef.current = time
      requestRef.current = requestAnimationFrame(animate)
    }

    requestRef.current = requestAnimationFrame(animate)
    return () => {
      if (requestRef.current) {
        cancelAnimationFrame(requestRef.current)
      }
    }
```

```
  }, [callback])
}
```

7.2 路径平滑算法

```
// 贝塞尔曲线平滑路径
const smoothPath = (points: Array<[number, number]>): string => {
  if (points.length < 2) return ''

  let path = `M ${points[0][0]} ${points[0][1]}`

  for (let i = 1; i < points.length - 1; i++) {
    const prev = points[i - 1]
    const curr = points[i]
    const next = points[i + 1]

    // 计算控制点
    const cp1x = curr[0] - (next[0] - prev[0]) * 0.2
    const cp1y = curr[1] - (next[1] - prev[1]) * 0.2
    const cp2x = curr[0] + (next[0] - prev[0]) * 0.2
    const cp2y = curr[1] + (next[1] - prev[1]) * 0.2

    path += ` C ${cp1x} ${cp1y}, ${cp2x} ${cp2y}, ${next[0]} ${next[1]}`
  }

  return path
}
```

7.3 碰撞检测

```
// AGV 碰撞检测
const detectCollision = (
  agv1: { position: [number, number]; radius: number },
  agv2: { position: [number, number]; radius: number }
): boolean => {
  const dx = agv1.position[0] - agv2.position[0]
  const dy = agv1.position[1] - agv2.position[1]
  const distance = Math.sqrt(dx * dx + dy * dy)

  return distance < (agv1.radius + agv2.radius)
}
```

八、风险与应对

风险	影响	应对措施
动画性能问题	页面卡顿	使用 Canvas 代替 SVG，减少 DOM 节点
路径计算复杂	加载时间长	后端预计算路径，前端只负责渲染
数据量过大	内存占用高	分页加载，虚拟滚动
浏览器兼容性	部分浏览器不支持	降级方案，使用静态图

九、后续扩展

9.1 3D 可视化

使用 Three.js 实现真正的 3D 数字孪生：

- 立体车间模型
- 真实设备模型
- 摄像机视角切换

9.2 实时数据同步

连接物理层实时数据：

- WebSocket 实时推送
- 设备状态同步
- 异常实时告警

9.3 VR/AR 支持

支持 VR 头显和 AR 设备：

- WebXR API 集成
- 沉浸式体验
- 手势交互

文档版本: 1.0
创建日期: 2025-01-XX
项目: 天工·弈控 - 天筹浑天可视化集成