# SmartScope 测试操作手册

## 📖 使用说明

本手册提供详细的测试操作步骤，确保每个测试都有明确的执行方法和验收标准。

## 🚀 快速开始

### 1. 环境准备

# 进入项目目录

cd /home/eddysun/App/Qt/SmartScope

# 给脚本执行权限

chmod +x 简化测试执行脚本.sh

# 运行自动化测试

./简化测试执行脚本.sh

### 2. 手动测试选择

如果需要单独执行某个测试，可以按照下面的详细步骤操作。

## 📋 详细测试步骤

### 第一阶段：基础功能测试

#### T001: 相机连接测试 ⏱️ 30分钟

测试目的: 验证系统能够正确识别和连接双目相机

前置条件:

* 双目相机已连接到USB端口
* 系统已安装相机驱动

详细步骤:

1. 检查硬件连接

# 检查USB设备

lsusb | grep -i camera

# 检查视频设备

ls /dev/video\*

预期结果: 应该看到 /dev/video0 和 /dev/video1

1. 运行相机测试程序

cd /home/eddysun/App/Qt/SmartScope

python3 camera\_test\_fixed.py

1. 观察测试输出

* 程序应该显示检测到的相机数量
* 应该能看到实时图像预览窗口
* 检查图像是否清晰，无花屏现象

1. 测试相机参数调节

* 在程序界面中调节亮度、对比度
* 观察图像变化是否及时响应
* 测试不同分辨率设置

验收标准:

* ✅ 检测到左右两个相机设备
* ✅ 图像显示清晰，帧率 ≥ 15fps
* ✅ 参数调节响应时间 < 2秒
* ✅ 无程序崩溃或异常

故障排除:

* 如果检测不到相机：检查USB连接，重新插拔
* 如果图像花屏：检查USB带宽，尝试降低分辨率
* 如果程序崩溃：检查权限，使用 sudo 运行

---

#### T002: 相机同步测试 ⏱️ 20分钟

测试目的: 验证双目相机的时间戳同步精度

详细步骤:

1. 启动同步测试

cd /home/eddysun/App/Qt/SmartScope/build

# 如果测试程序存在

./test\_camera\_sync --duration 60

# 如果不存在，使用Python脚本

cd ..

python3 -c "

import cv2

import time

cap0 = cv2.VideoCapture(0)

cap1 = cv2.VideoCapture(1)

sync\_errors = []

for i in range(100):

ret0, frame0 = cap0.read()

t0 = time.time()

ret1, frame1 = cap1.read()

t1 = time.time()

if ret0 and ret1:

sync\_diff = abs(t1 - t0) \* 1000 # 转换为毫秒

sync\_errors.append(sync\_diff)

print(f'Frame {i}: 同步差异 {sync\_diff:.2f}ms')

avg\_error = sum(sync\_errors) / len(sync\_errors)

print(f'平均同步误差: {avg\_error:.2f}ms')

print(f'最大同步误差: {max(sync\_errors):.2f}ms')

"

1. 分析同步数据

* 记录每帧的时间戳差异
* 计算平均同步误差
* 统计同步失败次数

验收标准:

* ✅ 平均同步误差 ≤ 5ms
* ✅ 最大同步误差 ≤ 10ms
* ✅ 同步成功率 ≥ 95%

---

#### T003: 配置文件测试 ⏱️ 15分钟

测试目的: 验证系统配置的加载、修改和保存功能

详细步骤:

1. 备份原配置

cd /home/eddysun/App/Qt/SmartScope

cp config.toml config.toml.backup

1. 测试配置加载

# 查看当前配置

cat config.toml

# 修改配置文件

sed -i 's/log\_level = "debug"/log\_level = "info"/' config.toml

# 验证修改

grep "log\_level" config.toml

1. 测试程序配置加载

# 启动程序，观察是否使用新配置

./SmartScopeQt --test-mode

# 检查日志级别是否改变

tail logs/app.log

1. 恢复配置

mv config.toml.backup config.toml

验收标准:

* ✅ 配置文件修改成功
* ✅ 程序正确加载新配置
* ✅ 错误配置有适当提示

---

#### T004: 立体匹配基础测试 ⏱️ 25分钟

测试目的: 验证立体匹配算法的基本功能

详细步骤:

1. 准备测试数据

cd /home/eddysun/App/Qt/SmartScope/reference\_code/lightstereo\_inference

# 检查测试数据是否存在

ls test\_data/ || echo "需要准备测试图像对"

1. 运行立体匹配

# 给脚本执行权限

chmod +x run\_example.sh

# 运行示例

./run\_example.sh

1. 检查输出结果

# 查看生成的文件

ls -la .png .jpg

# 如果有图像查看器，查看视差图

eog disparity.png 2>/dev/null || echo "视差图已生成"

1. 性能测试

# 测量处理时间

time ./run\_example.sh

验收标准:

* ✅ 成功生成视差图文件
* ✅ 视差图质量良好（无大片噪声）
* ✅ 处理时间 ≤ 500ms（640x480分辨率）

---

#### T005: 深度计算测试 ⏱️ 20分钟

测试目的: 验证深度图生成和深度值计算

详细步骤:

1. 运行深度计算

cd /home/eddysun/App/Qt/SmartScope

# 如果存在深度计算脚本

python3 test\_depth\_calculation.py

1. 如果脚本不存在，创建简单测试

cat > simple\_depth\_test.py << 'EOF'

import cv2

import numpy as np

# 创建模拟视差图

disparity = np.random.randint(0, 64, (480, 640), dtype=np.uint8)

# 模拟相机参数

focal\_length = 500 # 焦距

baseline = 60 # 基线距离(mm)

# 计算深度

depth = np.zeros\_like(disparity, dtype=np.float32)

mask = disparity > 0

depth[mask] = (focal\_length \* baseline) / disparity[mask]

print(f"深度范围: {depth[mask].min():.1f} - {depth[mask].max():.1f} mm")

print(f"平均深度: {depth[mask].mean():.1f} mm")

print("深度计算测试完成")

EOF

python3 simple\_depth\_test.py

验收标准:

* ✅ 深度计算程序正常运行
* ✅ 深度值在合理范围内（50-2000mm）
* ✅ 无计算错误或异常

---

#### T006: 用户界面基础测试 ⏱️ 30分钟

测试目的: 验证图形用户界面的基本功能

详细步骤:

1. 启动主程序

cd /home/eddysun/App/Qt/SmartScope/build

./SmartScopeQt

1. 界面功能测试

* 菜单栏测试: 点击每个菜单项，检查是否有响应
* 工具栏测试: 点击工具栏按钮，观察功能是否正常
* 页面切换: 测试不同页面间的切换
* 参数面板: 调节各种参数，观察界面更新

1. 响应性测试

* 快速点击多个按钮
* 拖拽窗口大小
* 最小化/最大化窗口

1. 错误处理测试

* 输入非法参数值
* 测试边界条件

验收标准:

* ✅ 界面正常显示，无布局错误
* ✅ 按钮响应时间 < 500ms
* ✅ 页面切换流畅
* ✅ 错误输入有适当提示

---

#### T007: 文件操作测试 ⏱️ 20分钟

测试目的: 验证文件的保存、加载和管理功能

详细步骤:

1. 创建测试目录

mkdir -p /tmp/smartscope\_test

cd /tmp/smartscope\_test

1. 测试图像保存

# 在主程序中保存图像，或使用命令行测试

python3 -c "

import cv2

import numpy as np

# 创建测试图像

img = np.random.randint(0, 255, (480, 640, 3), dtype=np.uint8)

# 保存不同格式

cv2.imwrite('test\_image.jpg', img)

cv2.imwrite('test\_image.png', img)

cv2.imwrite('test\_image.bmp', img)

print('图像保存测试完成')

"

1. 测试文件加载

# 验证文件是否正确保存

ls -la test\_image.\*

# 测试文件加载

python3 -c "

import cv2

# 加载图像

img\_jpg = cv2.imread('test\_image.jpg')

img\_png = cv2.imread('test\_image.png')

if img\_jpg is not None and img\_png is not None:

print('图像加载测试通过')

else:

print('图像加载测试失败')

"

1. 清理测试文件

rm -rf /tmp/smartscope\_test

验收标准:

* ✅ 支持多种图像格式保存（JPG, PNG, BMP）
* ✅ 文件保存完整，无损坏
* ✅ 文件加载正常显示

---

#### T008: 日志系统测试 ⏱️ 15分钟

测试目的: 验证日志记录和管理功能

详细步骤:

1. 检查日志目录

cd /home/eddysun/App/Qt/SmartScope

# 检查日志目录是否存在

ls -la logs/

# 如果不存在则创建

mkdir -p logs

1. 生成测试日志

# 启动程序生成日志

./SmartScopeQt --test-mode &

sleep 10

pkill SmartScopeQt

# 检查日志文件

ls -la logs/

1. 分析日志内容

# 查看日志内容

cat logs/app.log

# 检查日志格式

head -5 logs/app.log

# 检查错误日志

grep -i error logs/app.log || echo "无错误日志"

1. 测试日志轮转

# 创建大量日志测试轮转

for i in {1..1000}; do

echo "$(date): 测试日志条目 $i" >> logs/app.log

done

# 检查文件大小

ls -lh logs/app.log

验收标准:

* ✅ 日志文件正常生成
* ✅ 日志格式包含时间戳和级别
* ✅ 错误信息记录完整
* ✅ 日志文件大小控制合理

## 🎯 测试执行建议

### 1. 测试顺序

建议按照 T001 → T008 的顺序执行，因为后面的测试可能依赖前面的功能。

### 2. 时间安排

* 单人执行: 每个测试按预估时间执行
* 团队执行: 可以并行执行独立的测试项目
* 自动化执行: 使用提供的脚本可以大幅缩短时间

### 3. 问题记录

每个测试都应该记录：

* 执行时间
* 遇到的问题
* 解决方案
* 改进建议

### 4. 结果验证

* 所有 ✅ 标记的验收标准都必须满足
* 如果某项测试失败，应该分析原因并重新测试
* 记录测试环境和配置信息

## 📊 测试报告模板

测试项目: T001 - 相机连接测试

执行时间: 2024-08-06 14:30:00

执行人员: 张三

测试环境: Ubuntu 20.04, RK3588, 双目相机

测试结果:

✅ 检测到左右两个相机设备

✅ 图像显示清晰，帧率 18fps

✅ 参数调节响应时间 1.2秒

✅ 无程序崩溃或异常

问题记录:

* 初次启动时相机检测较慢，约5秒
* 建议增加检测进度提示

总体评价: 通过

这样的详细操作手册确保了每个测试都有明确的执行步骤和验收标准，提高了测试的可操作性和一致性。