在Linux层处理数据并推流到Android层的红外图像处理软件

使用手册

版本号	生成日期	作者	修订内容
v1.0	2024-04-09	肖劲涛	初始版本

1. 总体功能描述

《在Linux层处理数据并推流到Android层的红外图像处理软件》是一款创新性的软件平台，其核心设计目标是在GNU/Linux系统上实现红外图像的高效处理和推流至Android层，通过V4L2接口与RTSP协议的结合，为用户提供了一体化、灵活多样的红外图像处理方案。

1. 多摄像头数据并发输入：该平台支持连接多个摄像头，包括红外与可见光摄像头，实现数据并发输入。这为用户提供了更全面的图像信息，拓展了应用场景。
2. 高效数据处理：在Linux系统层面，通过V4L2接口实现对摄像头数据的处理，包括但不限于图像增强、滤波、温度计算等。平台通过优化算法，确保在数据处理过程中的高效性与准确性。
3. RTSP协议推流：经过数据处理后，平台利用RTSP协议将处理后的红外图像流进行推流。这使得用户可以远程实时查看处理后的图像，实现远程监控与控制。
4. Android端图像接收与显示：在Android层，用户可以通过专门的应用接收RTSP推流，并以类似相机的方式展现图像。这为用户提供了友好的操作界面，方便实时观察红外图像。
5. 灵活的应用适配性：平台设计灵活，适用于多个应用场景。不仅限于工业、电力、医疗等领域，还可以轻松应用于其他需要红外图像处理的领域，满足用户多样化的需求。
6. 实时性与稳定性：通过RTSP协议的使用，平台保证了数据传输的实时性，并通过系统优化保障了稳定性。用户可以在不同场景下获得高质量、低延迟的红外图像展示。
7. 用户友好的界面设计：Android端应用的设计注重用户体验，提供直观、简洁的界面，使用户能够轻松操作、查看并控制红外图像处理平台。

综合而言，《在Linux层处理数据并推流到Android层的红外图像处理软件》通过多方面的功能设计，为用户提供了一套完整而高效的红外图像处理方案，具有广泛的应用前景。

2. 运行环境

硬件要求

类别	基本要求
移植设备	近红外成像仪
测试设备	个人电脑

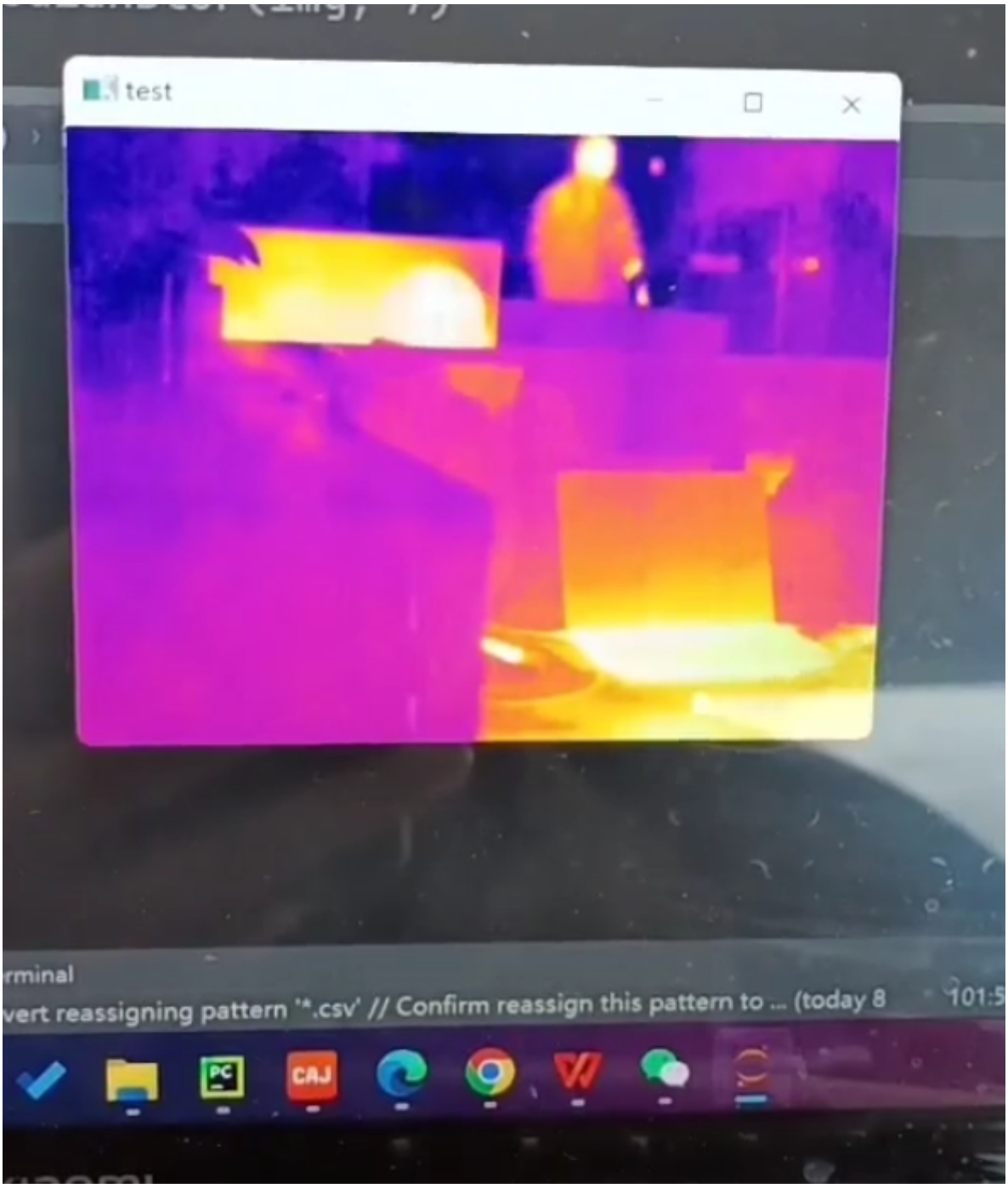
软件要求

类别	基本要求
目标系统	Android 11 with Kernel 4.19
测试系统	Windows 11

3. 编译环境

1. 目标编译器：aarch64-none-linux-gnu
2. 目标编译环境：Ubuntu16.04。
3. 测试编译器：MSVC 14.31
4. 测试编译环境：Windows 11

4. 软件测试效果



5. 软件具体描述

为了在我司的红外热像仪上实现该平台的功能，我们首先需要确认如下信息：

1. CPU的架构：ARMv8-a
2. Host的架构：X86_64
3. 可以使用的编译器版本：gcc-arm-10.3-2021.07-x86_64-aarch64-none-linux-gnu

5.1 配置FFmpeg

在确认好各项架构、版本之后，我们则首先静态编译FFmpeg。我们先`FFmpeg.org`获取FFmpeg，本平台使用的版本为4.3.1。之后我们在其`configure`文件中关注如下重点：

1. 配置相应的编译器、链接工具等。
2. 选择使用静态编译。
3. 开启network、tcp以及rtsp协议。

其完整配置如下：

```
#!/bin/bash
export TOOLS=/home/xjt/Gogs/gcc-arm-10.3-2021.07-x86_64-aarch64-none-linux-gnu
export SYSROOT=${TOOLS}/aarch64-none-linux-gnu/libc
# export PREFIX=./linux/arm64
export LD=${TOOLS}/bin/aarch64-none-linux-gnu-ld
export AR=${TOOLS}/bin/aarch64-none-linux-gnu-ar
export RANLIB=${TOOLS}/bin/aarch64-none-linux-gnu-ranlib

function build_lib
{
  ./configure \
  --disable-shared \
  --enable-static \
  --prefix=$PREFIX \
  --cross-prefix=${TOOLS}/bin/aarch64-linux-gnu- \
  --cc=${TOOLS}/bin/aarch64-none-linux-gnu-gcc \
  --nm=${TOOLS}/bin/aarch64-none-linux-gnu-g++ \
  --ld=${LD} \
  --ar=${AR} \
  --ranlib=${RANLIB} \
  --target-os=linux \
  --arch=arm64 \
  --sysroot=$SYSROOT \
  --enable-runtime-cpudetect \
  --enable-cross-compile \
  --enable-pic \
  --enable-gpl \
  --enable-nonfree \
  --enable-yasm \
  --enable-muxer=mpeg4 \
  --enable-muxer=rtsp \
  --enable-encoder=mpeg4 \
  --enable-decoder=mpeg4 \
  --enable-network \
  --enable-protocol=tcp \
  --enable-pthreads \
  --disable-ffmpeg \
  --disable-ffplay \
  --disable-ffprobe \
  --disable-avdevice \
  --disable-doc \
  --extra-ldflags="-L${SYSROOT}/libc/lib -L/home/xjt/Gogs/x264/install/lib -lc" \
  --extra-cflags="-I${SYSROOT}/libc/usr/include -I/home/xjt/Gogs/x264/install/include -Wfatal-errors -Wno-deprecated"
  # --enable-libx264 \
  # --extra-libs=-ldl
}
build_lib
```

5.2 创建应用程序

在生产FFmpeg静态库之后，我们可以按照如下方式创建我们的GNU/Linux程序。

```
.
├── build
│   ├── bin
│   ├── CMakeCache.txt
│   ├── CMakeFiles
│   ├── cmake_install.cmake
│   └── Makefile
├── CMakeLists.txt
├── lib
│   ├── ffmpeg
│   └── x264
└── src
    ├── CMakeLists.txt
    ├── main.cpp
    ├── v4l2_stream.c
    └── v4l2_stream.h
```

5.3 安卓系统适配

配置kernel

为了在安卓下使用`shm`，我们需要开启Sys V IPC功能，首先我们需要开启kernel的`general -> Sys V IPC`。

修改domain.te

之后我们将`./system/sepolicy/public/domain.te`中的`neverallow * *:{ shm sem msg msgq } *;`修改为`neverallow * *:{ sem msg msgq } *;`。这里根据需要移除对应的IPC选型，在编译时我遇到了报错，提示还需要保持`./system/sepolicy/prebuilts/api/30/public/domain.te`和上述文件一致。

修改check_vintf.cpp

安卓编译的时候还会有一个检查，以确保CONFIG_SYS_V_IPC设置为n，为了规避这项检查，我们需要修改`./system/libvintf/check_vintf.cpp`中的代码：

```
    if (compat.ok()) {
        std::cout << "COMPATIBLE" << std::endl;
        return EX_OK;
    }
    if (compat.error().code() == 0) {
        LOG(ERROR) << "files are incompatible: " << compat.error();
        std::cout << "INCOMPATIBLE" << std::endl;
        // return EX_DATAERR;
        return EX_OK;
    }
    LOG(ERROR) << strerror(compat.error().code()) << ": " << compat.error();
    return EX_SOFTWARE;
```

这里我们将`return EX_DATAERR;`修改为`return EX_OK;`，但是在修改的时候需要确保你的系统没有其他错误。建议先正常编译后，再开启Sys V IPC功能。