SD-Belt
User Manuel

SD Belt

(Scan & Detect Belt)

Group 15 a.k.a. “Belt Innovations”

01.06.2025

Table of Contents

Erkan Hocam before trying all modules one by one, please make sure you have up to date and correct repository and plase do not change anything in the repository.

Repository: https://github.com/Spexso/SD-Belt

1. Hardware Module User Manual

System Update and Compiler Installation

To ensure your system is up-to-date and has the necessary tools for compiling software:

- Update and upgrade the system: **sudo apt update && sudo apt upgrade -y**
- Install essential development tools: **sudo apt install build-essential -y**

Arduino CLI Setup

Install Arduino CLI to compile and upload code to the Arduino Uno:

- Install Arduino CLI: **curl -fsSL https://raw.githubusercontent.com/arduino/arduino-cli/master/install.sh | sh**
- Update core index: **arduino-cli core update-index**
- Install AVR board support: **arduino-cli core install arduino:avr**
- Install required libraries (e.g., Servo): **arduino-cli lib install Servo**

Check Arduino port:

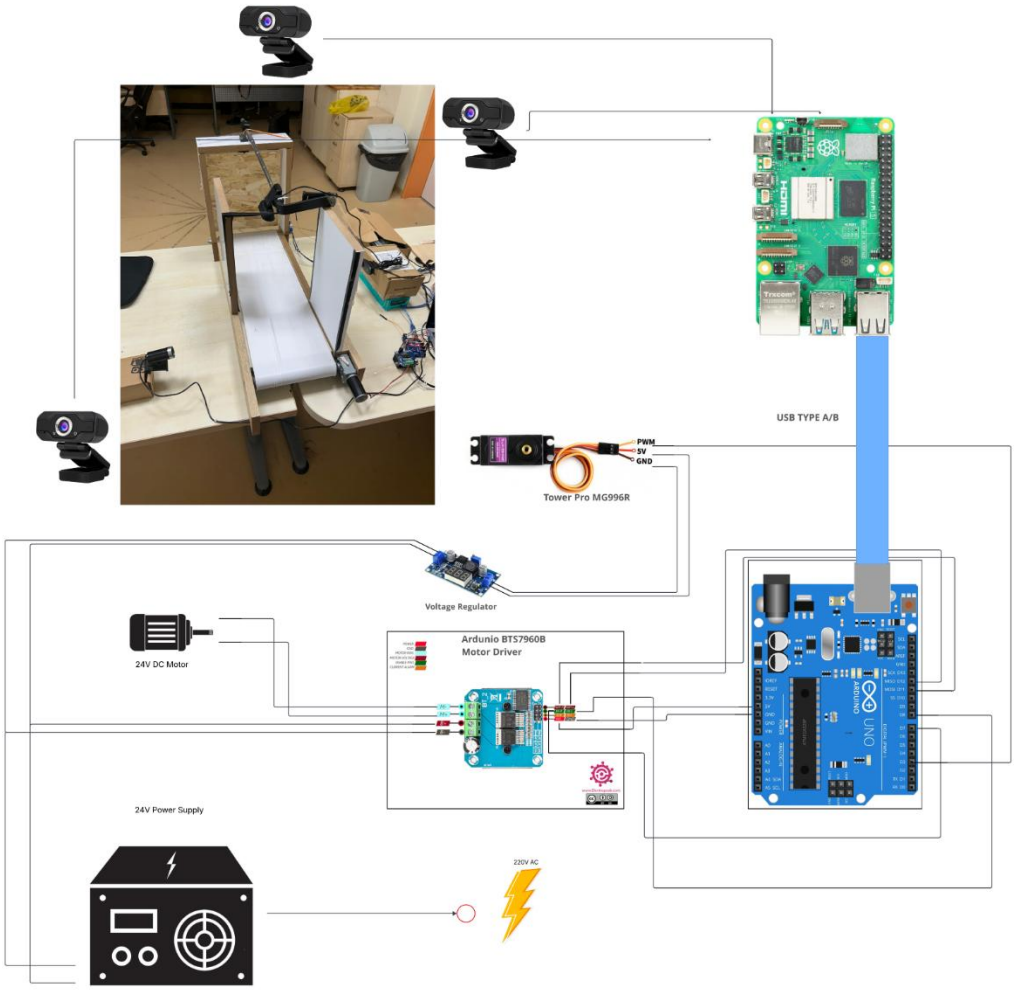
To determine the correct serial port the Arduino is connected to:

ls /dev/ttyUSB*

ls /dev/ttyACM*

All Connections:

All system connections, including pin details, are illustrated in the diagram.



2. Backend Module User Manual

System Requirements

- Java 17 or higher
- Maven 3.6+
- Docker and Docker Compose (for production deployment)
- Network access to port 6060 (application) and 8080 (system server)

Configuration Overview

The application uses the following key configurations:

- **Application Port**: 6060
- **Database**: H2 (development) or PostgreSQL (production)

Development Mode Setup

Prerequisites

Ensure Java 17+ and Maven are installed on your system.

Configuration

The application is pre-configured for development mode with H2 database. Key settings in application.properties:

spring.application.name=sdbelt

spring.datasource.url=jdbc:h2:mem:testdb

spring.h2.console.enabled=true

spring.h2.console.path=/h2-console

server.port=6060

system.server.host=172.20.10.4

Running in Development Mode

1. **Clean and Install Dependencies**
2. mvn clean install
3. **Run the Application**
4. java -jar target/sdbelt-*.jar
5. **Access Points**
    - Application: http://localhost:6060
    - H2 Database Console: http://localhost:6060/h2-console
        - JDBC URL: jdbc:h2:mem:testdb
        - Username: sa
        - Password: (leave empty)

Production Deployment with Docker Compose

Prerequisites

- Docker Engine 20.10+
- Docker Compose 2.0+

Setup Process

1. **Modify Application Configuration for Production**

Update your existing application.properties by changing these values:

Change database from H2 to PostgreSQL

spring.datasource.url=jdbc:postgresql://postgres:5432/sdbelt

spring.datasource.driverClassName=org.postgresql.Driver

spring.datasource.username=sdbelt_user

spring.datasource.password=your_secure_password

Change JPA dialect

spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect

```
# Disable H2 console
spring.h2.console.enabled=false

# Disable SQL logging for production
spring.jpa.show-sql=false

# Update server IP to your actual server IP
system.server.host=YOUR_SERVER_IP_HERE
```

---

2. **Docker Compose Configuration**

Create docker-compose.yml:

```
version: '3.8'
services:
 app:
  build: .
  ports:
   - "6060:6060"
  depends_on:
   - postgres
  networks:
   - sdbelt-network

 postgres:
  image: postgres:15
  environment:
```

```
    POSTGRES_DB: sdbelt
    POSTGRES_USER: sdbelt_user
    POSTGRES_PASSWORD: your_secure_password
  volumes:
    - postgres_data:/var/lib/postgresql/data
  networks:
    - sdbelt-network

volumes:
  postgres_data:

networks:
  sdbelt-network:
    driver: bridge
```

3. **Build and Deploy**
4. # Build the application
5. mvn clean install
6.
7. # Start services
8. docker-compose up -d
9.
10. # View logs
11. docker-compose logs -f app
12. **Access Application**
    - Application: http://localhost:6060
    - Database: Accessible only within Docker network

Production Management

- **Stop Services**: docker-compose down
- **Update Application**: Rebuild with docker-compose up --build -d
- **Database Backup**: Use standard PostgreSQL backup tools
- **Monitoring**: Check logs with docker-compose logs

Network Configuration

System Server Integration

The application connects to a system server (configured in system.server.host). Ensure:

- Network connectivity to your configured server IP
- Port 8080 is accessible on the target server
- Firewall rules allow outbound connections

Port Configuration

- **Application Port**: 6060 (configurable via server.port)
- **System Server Port**: 8080 (fixed)
- **Database Ports**: 5432 (PostgreSQL, internal to Docker network)

Troubleshooting

Common Issues

1. **Port Conflicts**: Change server.port if 6060 is occupied
2. **Database Connection**: Verify PostgreSQL container is running
3. **System Server Unreachable**: Check network connectivity to your configured server IP:8080
4. **H2 Console Access**: Ensure spring.h2.console.enabled=true in development

Health Checks

Monitor application health at: http://localhost:6060/actuator/health (if Spring Actuator is enabled)

3. Desktop Application Module User Manual

Prerequisites

System Requirements

• Windows: 10 or later
• Linux: Ubuntu 20.04 or later

Required Software

• Qt Creator 6.5 or later
• Qt Framework 6.5 or later
• CMake (optional but recommended)
• GCC (Linux) / MinGW (Windows)

Required Qt Modules

• Qt6Core – Core functionality
• Qt6Gui – GUI rendering
• Qt6Widgets – Widget toolkit
• Qt6Network – HTTP & UDP communication
• Qt6Svg – For icon rendering
• Qt6SerialPort (optional)
• Qt6Concurrent (optional)

Development Environment Setup

1. Install Qt Creator and Qt Framework

1. Download Qt Online Installer
2. Run installer and select:

1. Qt 6.5 or latest LTS
2. Qt Creator
3. Desktop components"
4. Ensure necessary modules are checked

2. Install CMake (Optional)

1. Windows: Download CMake from official site
2. Linux: Run: sudo apt install cmake
3. sudo apt install build-essential qt6-base-dev libqt6network6

Configure Backend Connection

IMPORTANT: Before building the application, you must configure the backend server IP address.

1. **Open the project** in Qt Creator
2. **Navigate to** `Globals.h`
3. **Find the** `BackendAddress` and `RaspListenerAddress`

```
#define PercentSign "%"
#define CelciusSign "C°"
#define BackendAddress "http://192.168.18.249:6060/api/v1"  // Change this 192.168.18.249 part to your backend ip address
#define BackendPort "6060"
#define SystemStopPoint "/system/stop"
#define SystemSpeedPoint "/system/speed"
#define SystemReversePoint "/system/reverse"
#define SystemNotifyPoint "/system/notify"
#define SystemLogs "/system/logs"

#define AccessSystemInfoUrl "/system/info"
#define AccessLogsUrl "/scans"


// Camera Receiver
#define WIDGET_H 240
#define WIDGET_W 320
#define CAM_COUNT 3


// MainWindow
#define ProjectName "SD-Belt"
#define RaspListenerAddress "http://192.168.91.204" // Change the 192.168.91.204 part to your rasberry pi ip address

#endif // GLOBALS_H
```

4. **Replace the IP address** `172.168.18.249` with your backend server's IP address if you are going to use the system in local network.

5. **Keep the port** `:6060/api/v1` unless your backend uses a different port.
6. **Replace the IP address** `172.168.91.204` with your Raspberry Pi IP address for live camera stream in desktop application.
7. **Save the file** before building.

Desktop Deployment

A. Open Project in Qt Creator

1. Launch Qt Creator
2. Go to File → Open File or Project
3. Choose CMakeLists.txt or SD-Belt.pro

B. Configure Build Kit

1. Select appropriate desktop kit (GCC or MinGW)
2. Verify modules are detected
3. Click Configure Project

C. Build Project

1. Go to Build → Clean All
2. Go to Build → Build Project
3. Ensure there are no compile-time errors

D. Run Project

1. Click Run
2. Make sure backend and Raspberry Pi are active
3. Desktop GUI should launch with all functional panels

4. Mobile Application User Manual

Prerequisites

System Requirements

- **macOS**: 10.15+ (for iOS development)
- **Windows**: 10+ (Android only)
- **Linux**: Ubuntu 18.04+ (Android only)

Required Software

- **Qt Creator** 6.9.0
- **Qt Framework** 6.9.0
- **Android Studio** (for Android development)
- **Xcode** (for iOS development - macOS only)

Development Environment Setup

1. Install Qt Creator and Qt Framework

1. Download Qt Online Installer
2. Run installer and select:
    - Qt 6.9.0 (or latest LTS)
    - Qt Creator
    - Android components (if targeting Android)
    - iOS components (if targeting iOS)

2. Install Android Studio (for Android)

1. Download Android Studio
2. Install with default settings
3. Open Android Studio and complete initial setup
4. Install required SDK components:
    - Android SDK Platform 28+

- Android SDK Build-Tools
- Android NDK
- Android SDK Platform-Tools

3. Install Xcode (for iOS - macOS only)

1. Install from Mac App Store
2. Launch Xcode and accept license agreements
3. Install additional components when prompted

Configure Backend Connection

IMPORTANT: Before building the application, you must configure the backend server IP address.

8. **Open the project** in Qt Creator
9. **Navigate to** apiservice.cpp file
10. **Locate the constructor** around line 14-20:

```cpp
ApiService::ApiService(QObject *parent) : QObject(parent)
{
    m_networkManager = new QNetworkAccessManager(this);
    //m_baseUrl = "https://sdbelt.devodev.online/api/v1";
    m_baseUrl = "http://172.20.10.6:6060/api/v1";  // ← Change this IP
    qDebug() << "ApiService initialized. Base URL:" << m_baseUrl;
}
```

11. **Replace the IP address** 172.20.10.6 with your backend server's IP address if you are going to use the system in local network.
12. **Keep the port** :6060/api/v1 unless your backend uses a different port.
13. **Save the file** before building.

Android Deployment

A. Configure Qt Creator for Android

1. **Open Qt Creator**
2. Go to **Preferences** → **Devices** → **Android**
3. Set paths (Qt Creator usually auto-detects):
    - **Android SDK Location**: /Users/[username]/Library/Android/sdk
    - **Android NDK Location**: /Users/[username]/Library/Android/sdk/ndk/[version]
    - **Java Location**: /usr/bin/java or auto-detected path

B. Prepare Android Device

Enable Developer Mode:

1. Go to **Settings** → **About Phone**
2. Tap **Build Number** 7 times rapidly
3. Return to main Settings
4. Find **Developer Options** (now it should be visible)

Enable USB Debugging:

1. In **Developer Options**, enable:
    - **USB Debugging**
    - **Install via USB**
    - **USB debugging (Security settings)** (if available)

Connect Device:

1. Connect Android device via USB cable
2. When prompted on device, tap **"Always allow from this computer"**
3. Select **"File Transfer"** or **"MTP"** mode (not charging only)

C. Deploy to Android Device

1. **Open Project** in Qt Creator
2. **Select Android Kit** from kit selector (top left)
3. **Build** → **Clean All**

4. **Build** → **Build Project**
5. **Run** → **Run on Android Device**

---

iOS Deployment

A. Configure Development Environment

1. **Install Xcode** from Mac App Store
2. **Open Xcode** and accept license agreements
3. **Sign in** with Apple ID (Xcode → Preferences → Accounts)

B. Prepare iOS Device

1. **Enable Developer Mode**:
    - Connect device to Mac
    - Open Xcode → Window → Devices and Simulators
    - Select your device and click "Use for Development"
2. **Trust Computer**:
    - When prompted on iOS device, tap "Trust This Computer"
    - Enter device passcode

C. Configure Qt Creator for iOS

1. **Qt Creator** → **Preferences** → **Devices** → **iOS**
2. Verify Xcode path is detected
3. Ensure iOS SDK is available

D. Deploy to iOS Device

1. **Select iOS Kit** in Qt Creator
2. **Build** → **Clean All**
3. **Build** → **Build Project**
4. **Connect iOS device** via USB
5. **Run** → **Run on iOS Device**

5.AI & Image Processing Module User Manual

Data Preparation & Training Scripts

This folder contains helper scripts and a notebook to prepare, analyze, and train your dataset and model. Each script is standalone and focuses on a specific preprocessing or evaluation task.

Table of Contents

Prerequisites

- Python 3.7+
- A working virtual environment (recommended)
- Required Python packages (see Dependencies)

Scripts Overview

Script	Description
add_false_samples.py	Augment minority classes by duplicating or generating false samples to balance dataset classes.
balance_count.py	Analyze class distribution and output target counts for balancing or sampling purposes.
compute_prune_targets.py	Calculate which samples or channels to prune based on class frequencies or model metrics.

Script	Description
count_class.py	Count the number of samples per class in your dataset directory.
count_null.py	Scan annotation files and report entries with missing or null labels.
delete_null.py	Remove samples or annotation entries that have null or invalid labels.
remap_labels.py	Remap label values according to a provided mapping (e.g., merge classes or standardize labels).
inference_test.py	Run inference on a set of images or videos, and report metrics (accuracy, speed).
train.ipynb	Jupyter notebook demonstrating end-to-end model training and evaluation workflow.

Usage

Activate your virtual environment and install dependencies:

```
python -m venv venv
source venv/bin/activate    # macOS/Linux
# .\venv\Scripts\activate   # Windows
pip install -r requirements.txt
```

Running a Script

Each script provides a --help flag showing its arguments:

```
python add_false_samples.py --help
python balance_count.py --help
```

Examples

 Count samples per class

```
python count_class.py \
```

```
  --data-dir /path/to/images \
  --output class_counts.csv
```

Remove null annotations

```
python delete_null.py \
  --annotations /path/to/annotations.json \
  --output cleaned_annotations.json
```

Remap labels

```
python remap_labels.py \
  --input labels.csv \
  --mapping label_map.yaml \
  --output remapped_labels.csv
```

Run inference test

```
python inference_test.py \
  --model checkpoint.pt \
  --input images/ \
  --output results.json
```

Open training notebook

```
jupyter lab train.ipynb
```

Dependencies

Managed via requirements.txt (create one if missing):

numpy

pandas

argparse

torch

torchvision

scikit-learn

pyyaml

opencv-python

jupyter

Install with:

pip install -r requirements.txt

This is the report of implementing 'h8l architecture .hef model running on hailo accelerator chip'

repo link : "https://github.com/Spexso/SD-Belt/tree/main/AI"

**SOME NOTES

repo suggest installing libboost related packages like this:

"apt install -y libboost-format-dev libboost-program-options-dev"

but this after

"make -j$(nproc)"

we got this error:

"fatal error: boost/format.hpp no such file or directory"

it turns out desired file is in core libboost-dev

so we corrected way of installing libraries:

"sudo apt install -y libboost-dev libboost-program-options-dev"

!!! this is not tested but it is recommended to use absolute path

give permission to download_resources.sh file for best practice

before "./download_resources.sh"

"chmod -x download_resources.sh"

instead of running :

"./build.sh"

"mkdir -p build && cd build"

"cmake .."

"make -j$(nproc)"

example run (mind the '=' symbol and make sure there is no gap between)

"./obj_det -hef=/home/sd_belt/object_detection_example/Hailo-Application-Code-Examples/runtime/cpp/object_detection/yolov8n.hef \

-input=/home/sd_belt/object_detection_example/Hailo-Application-Code-Examples/runtime/cpp/object_detection/bus.jpg"