

iQ9 Factory 4 Line Defect Detection Demo - Provisioning



Contents

Contents
Required Parts
Overview
File Repository
Required Software
Provisioning the iQ9 RB8
Flashing the Image
Prepare Qt Responsive SCADA Application
Setting up the Cameras
Networking
Chroma DB
MQTT
MQTT Broker
MQTT Client - Predictive Maintenance
RTSP
RTSP Server
RTSP GStreamer Config
Predictive Maintenance Sensor Fusion
Using the Demo
Appendix
QT Creator
System Diagram for Embedded World 2026
Using a different Model
Trouble Shooting

Required Parts

Quantity	Part	Link if applicable
1	iQ9 RB8	
4	TIS Camera	
2	1080p Display	
1	PoE Switch	<u>TP-Link LS108GP</u>
1	USB C Hub	<u>FORIDA USB C Hub-8 Port USB HUB</u>
1	USB Webcam	<u>Logitech HD Pro Webcam C920</u>
1	Ubuntu Laptop/Machine	
1	USB C Cable	Provided in RB8 Kit

1	USB Micro Cable	Provided in RB8 Kit
4	Camera Mounting Arm	Small Rig Arm
4	Lighting Mounting Arm	
4	Arduino BLE Nano 33	Amazon Link
4	INA219 Current Sensor	INA219
2	Mini Display Port to HMDI	BENFEI Mini DP to HDMI Note: It was difficult to find a compatible adapter. Order this specific model if possible
4	USB Lighting Ring	
6	Cat 5 Ethernet Cables	
1	Keyboard and Mouse	
4	Conveyor Belt	20 Inch Desktop Conveyor
20	Nespresso Pods Gold	Volluto Pods
1	High Vis Jacket	
1	Hard Hat	
	Re-useable Cable Ties	
4	USB to UART Adapter	Amazon Link
4	Custom Bread Board	This was specially created for EW, see wiring diagram in the doc if you wish to reproduce
4	Housing Box	IP55 Junction Box

Overview

The Responsive SCADA Demo was created in collaboration with the Qt Company. We have wrapped a number of AI applications into an interactive dashboard with a 3D Digital Twin model of a factory floor.

The Interactive GUI displays four separate pages of the factory line, each of which can display a live inference pipeline from a GigE Camera and plot a graph of sensor telemetry from the Arduino Nano.

<https://www.youtube.com/watch?v=L6RTqDvjafo>

Demo Resources:

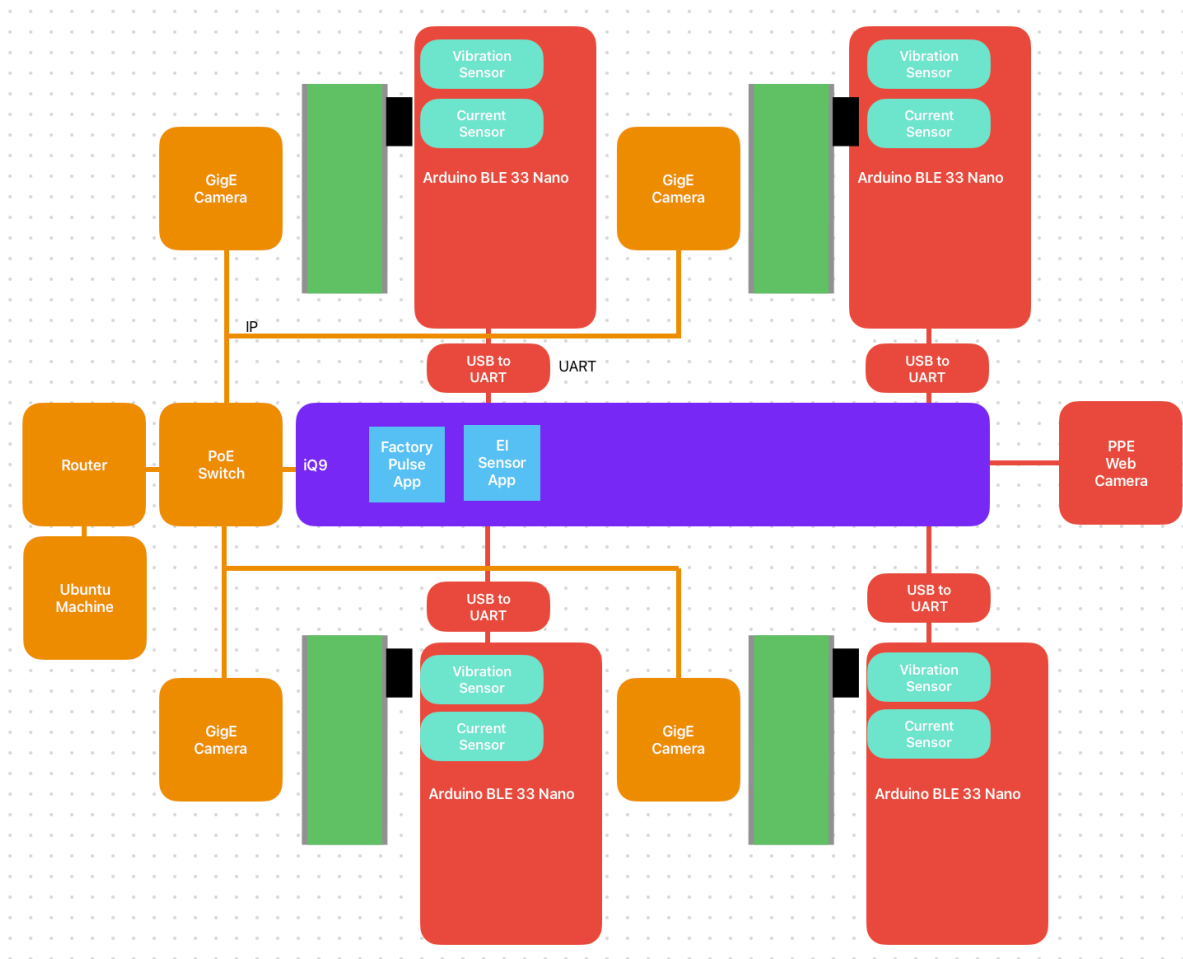
- Defect Detection with YOLO Pro Coffee Pods - [Studio Project](#)
- LLM with RAG - [Llama 3.2](#)
- RAG - [BGE-large](#)
- [Optional] Sensor Fusion Predictive Maintenance - [Studio Project](#) and [GitHub](#)

Below shows an illustration of the demo parts and how they are combined together. The iQ9 hosts two applications; The Factory Pulse Application built by Qt and the EI Sensor App built by Edge Impulse for the predictive maintenance element.

The Inspection Cameras are all GigE IP based cameras. The camera stream can either be delivered directly through the tcam-bin library or through an RTSP stream, both configs are provided in the instructions below.

Communication with the Arduino Nano for the predictive maintenance feature is via USB to UART adapters. The Current and Accelerometers are I2C based and communicate with the Arduino Nano directly.

The PPE Detection makes use of a USB Web cam and is connected to the iQ9 via the USB hub.



NOTE:

Only operate the motor in the forwards direction. Reversing the motor will break the INA219 Current Sensor. Forward is mark by the '||' symbol on the power cable.



File Repository

Some key files are stored in the google drive below. Please check you have the correct permissions to access this folder. If you do not have access please

contact markwaters@edgeimpulse.com

[Google Drive](#)

Required Software

Before progressing with this guide you will need to some software tools:

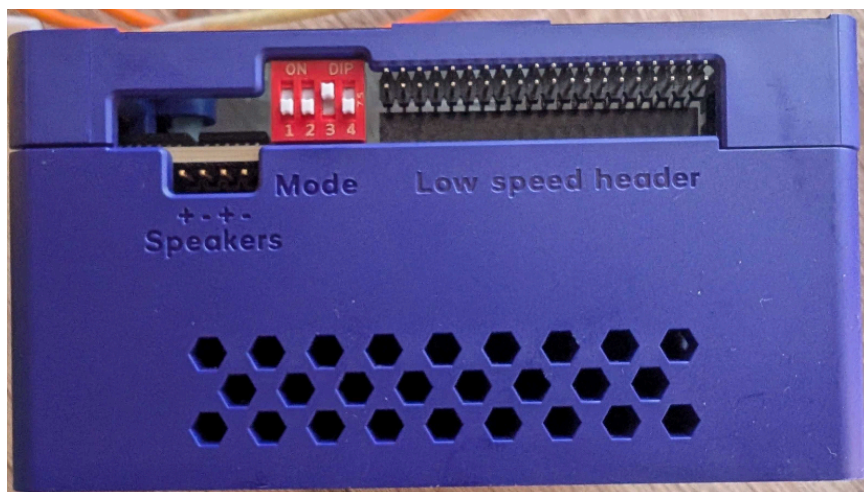
- QDL Tool - [Link](#)
- TIS Camera Software (Linux Only) - [Link](#)
 - The Debian Installation works with a Rubik Pi if you have one available [Link](#)

Provisioning the iQ9 RB8

This section will describe the provision method to prepare the iQ9 for this demo. This will flash a new image to the iQ9, if there are any existing work or projects on your device you will need to back these up now.

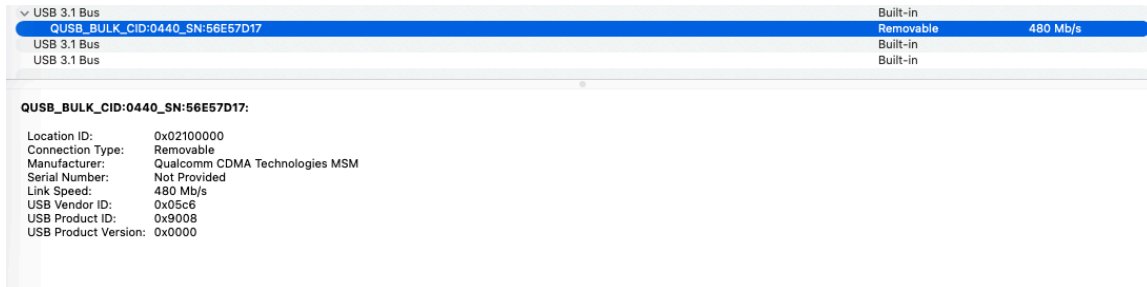
Flashing the Image

1. Connect the board to power
2. Flip DIP switch 3 to the up position to enable the EDL mode



1. Connect USB0 (Labelled ADB) to the host computer with a USB C cable
2. Power on the device with the on board power switch
3. Verify the device is in EDL mode

- a. This can be done on Mac by going to "About this Mac" >> "More Info" >> "General" >> "System Report" >> "Hardware" >> "USB"
The device should be listed under USB 3.1



4. Flash the device image

- a. Download the qcom-qt-image-qcs9075-iq-9075-evk.rootfs.tar.gz file from the shared google drive in the Images folder.

```
tar -zxvf qcom-qt-image-qcs9075-iq-9075-evk.rootfs.tar.gz
```

- b. **NOTE: If this is the first time the board is being flashed you will need to provision the UFS, Flash the CDT and the SAIL. If this is a re-flash you can skip this step and proceed to point c.**

i. UFS Provisioning

Download and Extract the UFS file from: [Here](#)

Flash with this command:

```
/path/to/qdl --storage ufs prog_firehose_ddr.elf provision_1_2.xml
```

ii. CDT

Download and Extract the CDT file from: [Here](#)

Flash with this command:

```
/path/to/qdl prog_firehose_ddr.elf rawprogram3.xml patch3.xml
```

iii. SAIL

The SAIL file is inside the QBSP folder which was extracted in the previous step

Flash with this command:

```
cd <Qt version folder>/qcom/qcs9075-iq-9075-evk/images/qcom-qt-image-qcs9075-  
iq-9075-evk.rootfs/sail_nor/  
/path/to/mdl --storage spinor prog_firehose_ddr.elf rawprogram0.xml patch0.xml  
l
```

- c. Use the command below to flash the QBSP Image

```
cd <image folder>/qcom-qt-image-qcs9075-iq-9075-evk.r  
ootfs/  
  
<path to>/mdl --storage ufs prog_firehose_ddr.elf raw  
program*.xml patch*.xml
```

5. Verify the Image by checking the board's debug connection

- a. Turn off the device
- b. Flip DIP switch 3 to the down position to disable EDL mode
- c. Connect a USB Micro cable to the debug port on the RB8
- d. Turn the device back on
- e. Connect to the Serial Debug port
 - i. `sudo minicom -D /dev/ttyUSB1`
 - ii. On mac the device will appear as a tty.usbserial device. Four devices will be listed, connect to the one ending in '1' e.g. minicom - D /dev/tty.usbserial-NNPMP51H004C1
- f. Log in with
user: root
password: oelinux123

▼ Prepare Qt Responsive SCADA Application

The Responsive Qt Application can be deployed onto the iQ9 RB8 in two ways; building from source and installing using the Qt Creator App, or using a precompiled app and an image overlay. The precompiled app with image overlay is the simpler method of deployment and will be detailed here.

- Download the Qt Overlay from the [google drive](#) folder
- Transfer to the iQ9 over a network using scp. If you are not able to use the Ethernet port for networking, please see the Networking session to

see instructions on how to connect via wifi.

- Copy the tar file across using ssh

```
scp target-overlay.tar.gz root@192.168.0.144:/root/.
```

- Log into the device using ssh
- Remount the file system

```
mount -o remount,rw /usr
```

- Extract the overlay on the device to merge the root file system. Run this in the base directory:

```
cd /  
tar xvf root/target-overlay.tar.gz
```

- Reboot the the device
- The Application will launch automatically via a **systemd** service called **demolauncher**
- You can stop and start the app with **systemctl**

```
systemctl stop demolauncher  
systemctl start demolauncher
```

- **NOTE:** The Application will launcher two separate windows. If only one screen is connected only one window will appear. Your testing will be severely limited with only one screen connected.
- By default the application will launch the config.json shipped with the overlay. This will not make use of the TIS camera for visual inspection. The config file must be updated to use a live camera. Without a TIS camera the demo can load a pre-recorded video.
- The systemd service make use of a number of environment variables which are exported in the service file
- If you wish to run the app in a standalone shell you will need to export the same environment variables as in the service:

```

[Service]
Type=idle
User=root
; StandardOutput=null
; StandardError=null
; LogLevelMax=error
; SyslogLevel=error
Environment="QMONITOR_BACKEND_LIB_PATH=/var/QualcommProfil
iler/libs/backends"
Environment="XDG_RUNTIME_DIR=/dev/socket/weston"
Environment="WAYLAND_DISPLAY=wayland-1"
Environment="QT_QPA_PLATFORM=wayland-egl"
Environment="PATH=/var/QualcommProfiler/bins:/usr/local/
sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
Environment="LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/var/Qualc
ommProfiler/libs"
Environment="QUICKMQTT_URL=http://192.168.0.133:1883/"
WorkingDirectory=/home/jari
ExecStartPre=/bin/sleep 1
ExecStart=-/usr/bin/sh -c "/home/jari/Responsive_ScadaAp
p"
Restart=always
RestartSec=1

[Install]
WantedBy=multi-user.target

```

▼ Setting up the Cameras

The cameras are set up with the TIS camera software listed in the software requirements above. You will now need to use the linux machine with the TIS camera software installed.

To be able to set up the Responsive SCADA app with the TIS Camera, you need to find the device serial number with the following command:

```
tcam-ctrl --list
```

It is also necessary to focus and adjust the cameras to suit your demo set up. This can be achieved using the tcam-capture application included with the TIS software.

- Launch the tcam-capture app

```
tcam-capture
```

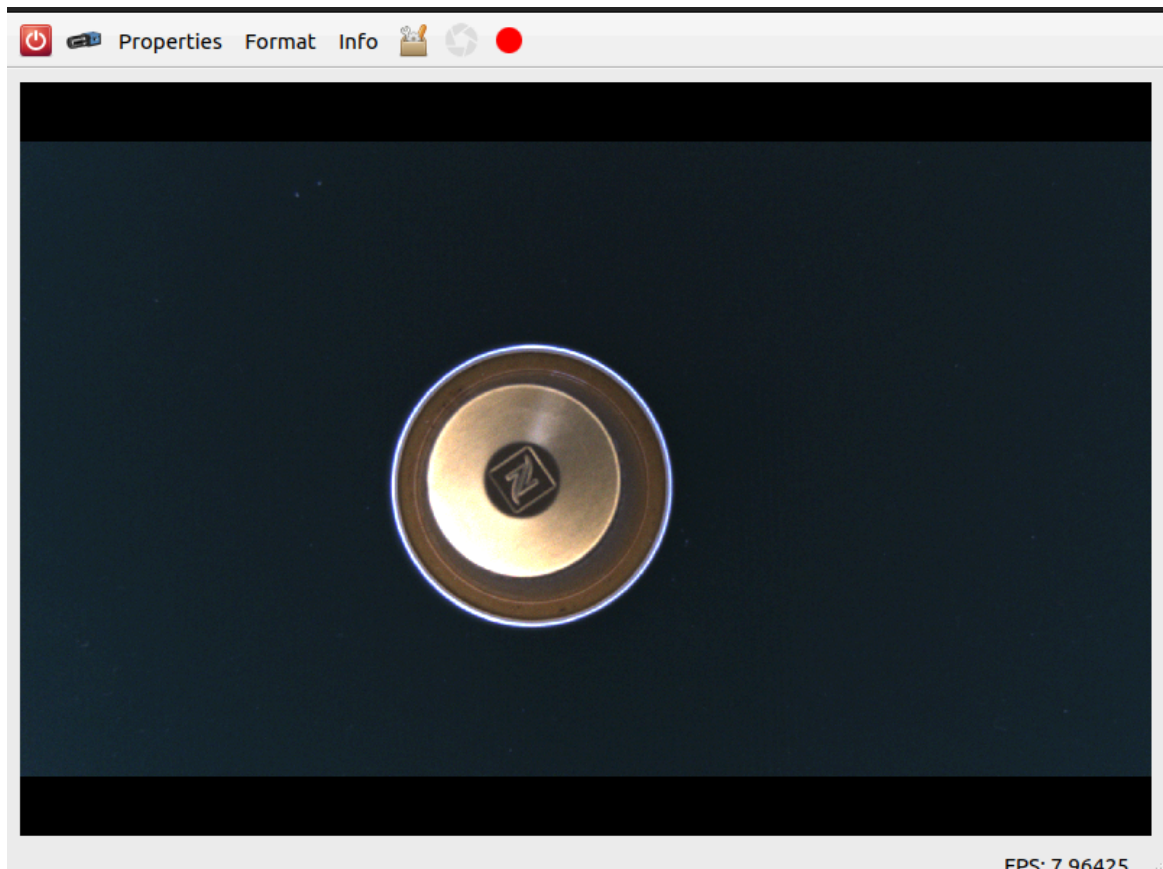
- The application should show the video stream output. If the video does not show, it is likely there is an issue with the networking settings of either the host computer or the camera
 - Check the ip address of the host computer
 - Force the camera to take an ip address in the same subnet. **Note** this should be on the same subnet as the iQ9 is.

```
tcam-gigetool rescue --ip 192.168.0.100 --netmask 255.255.255.0 --gateway 192.168.0.1 [camera-serial]
```

- Save the ip config to the camera

```
tcam-gigetool set --ip 192.168.0.100 --netmask 255.255.255.0 --gateway 192.168.0.1 --mode static [camera-serial]
```

- Reboot the camera
- Relaunch tcam-capture to test if the video stream can now be viewed



- You will need to match the camera view as close to the above picture as you can. The model was trained with data with a similar view, changing this perspective will decrease model performance
- **Focus the Camera** - Go to Lens, adjust the zoom to match the above picture. Click on Auto Focus and select 'once'
- **Set White Balance** - Go the 'Color' tab and select 'Once' from the Auto White Balance menu
- **Set Exposure** - From the Exposure Auto, first select 'Continuous' and then select 'Off'
- **Set Gain** - on the same tab, select 'Continuous' from the Gain Auto menu and then select 'Off'
- Once the camera is focused and exposed correctly, save the config and close the app
- Keep the camera on and start launch the SCADA app.
- **NOTE:** The settings for the camera do not persist with a power cycle, keep the cameras powered throughout the demo. You will have to repeat this process if the cameras lose power.

The application will look for a config.json file in the same directory as the binary. For the overlay pre built app this is located in /home/jari/

Edit the config file to identify the camera and pipe line option.

Example config files can be found in the [google drive](#).

For Example this config could be used for 4 TIS cameras. **NOTE:** Serial numbers for the cameras will differ. Use the *tcam-ctrl* *—list* function to find your own.

You can also rotate the video output by changing the 'videoflip' method to clockwise or counterclockwise.

```
{
  "defect-detection-source-line1": "tcambin serial=066
21138 ! capsfilter caps=video/x-raw,format=BGRx,width=10
80,height=960,framerate=24/1 ! queue ! videoconvert ! vi
deo/x-raw,format=RGB ! queue ! videoflip method=none ! e
dgeimpulsevideoinfer ! edgeimpulseoverlay ! videoconver
t",
  "defect-detection-source-line2": "tcambin serial=066
21135 ! capsfilter caps=video/x-raw,format=BGRx,width=10
80,height=960,framerate=24/1 ! queue ! videoconvert ! vi
deo/x-raw,format=RGB ! queue ! videoflip method=none ! e
dgeimpulsevideoinfer ! edgeimpulseoverlay ! videoconver
t",
  "defect-detection-source-line3": "tcambin serial=066
21133 ! capsfilter caps=video/x-raw,format=BGRx,width=10
80,height=960,framerate=24/1 ! queue ! videoconvert ! vi
deo/x-raw,format=RGB ! queue ! videoflip method=none ! e
dgeimpulsevideoinfer ! edgeimpulseoverlay ! videoconver
t",
  "defect-detection-source-line4": "tcambin serial=066
21160 ! capsfilter caps=video/x-raw,format=BGRx,width=10
80,height=960,framerate=24/1 ! queue ! videoconvert ! vi
deo/x-raw,format=RGB ! queue ! videoflip method=none ! e
dgeimpulsevideoinfer ! edgeimpulseoverlay ! videoconver
t",
  "demo-mode-timer": 10,
  "restricted-area-path": [
```

```

    {
        "x": 180,
        "y": 260
    },
    {
        "x": 480,
        "y": 260
    },
    {
        "x": 480,
        "y": 480
    },
    {
        "x": 180,
        "y": 480
    }
]
}

```

▼ Web Cam

The web cam will be used for the PPE detection. When connected the device should appear as /dev/video2

The app will automatically look for a camera device at this location and no further config is required.

The model for ppe detection is found in /etc/models/

Networking

To set up Wi-Fi use the following commands:

```

wpa_passphrase <SSID name> <password> > /etc/wpa_supplicant.conf
wpa_supplicant -B -iwlan0 -c /etc/wpa_supplicant.conf -Dnl80211,wext
dhcpcd wlan0

```

Note : These changes will not be permanent. I recommend using the Ethernet port when possible.

To enable DNS lookup use the following command:

```
ln -sf /etc/resolv-conf systemd /etc/resolv.conf
```

This will be required to install python packages for MQTT

Chroma DB

Chroma DB is used for the RAG element of the demo.

- Install chromadb python package

```
pip3 install chromadb
```

MQTT

MQTT is used a number of different aspects of the Responsive SCADA app.

Integration with LLM

Integration with the iQ6 HMI Demo.

Integration with Sensor Fusion Model Predictive Maintenance Model.

MQTT Broker

Install the Mosquitto MQTT Broker service to Ubuntu PC/Laptop to host the local MQTT service.

```
sudo apt-add-repository ppa:mosquitto-dev/mosquitto-ppa  
sudo apt-get update  
sudo apt install mosquitto
```

Configure the broker to accept connections from external networks in
/etc/mosquitto/mosquitto.conf

```
allow_anonymous true  
listener 1883
```

Restart the service

```
service mosquitto restart
```

MQTT Client - Predictive Maintenance

To read the sensor readings from the Arduino Nano devices a standalone app is hosted on the iQ9. This app will search for USB devices listed a config file and will read the sensor values sent across the bus. Those readings will be parsed and packaged into MQTT messages and sent to the broker.

The Responsive SCADA Qt App is subscribed to these topics and will plot the sensor readings on a small dashboard.

Installing Steps:

- scp the conveyor-monitor project to the iQ9

```
scp -r iq9-conveyor-monitor root@192.168.0.101:/home/
```

- Install the requirements. If this fails, it could be because you haven't mounted the file system or DNS hasn't been enabled.

```
pip3 install -r requirements.txt
```

- Create a config.json file to match the set up you have with the correct broker address and number of lines. Each line should have an entry under 'devices' with the corresponding USB device name.

```
{
  "mqtt": {
    "host": "192.168.0.133",
    "port": 1883,
    "topic_prefix": "EdgeImpulse/Workshop/Dashboard"
  },
  "devices": [
    { "port": "/dev/ttyUSB0", "name": "Line1", "baud":
115200 },
  ]
}
```

- Update the MQTT namespace config file found under /root/.config/The Qt Company/FactoryPulse.conf

```
/root/.config/The Qt Company/FactoryPulse.conf
```

- Here's an example config that be used for 4 lines which will all be under an EdgeImpulse namespace

```
[FactoryPulse]
companies="Qt,Qualcomm,EdgeImpulse"
kpiNs=EdgeImpulse
line1Ns=EdgeImpulse
line2Ns=EdgeImpulse
line3Ns=EdgeImpulse
line4Ns=EdgeImpulse
mqttBrokerIp=192.168.0.133
mqttNamespace=EdgeImpulse
```

- Move the service file to the systemd folder, enable and start the service

```
cp conveyor-monitor.service /etc/systemd/system/
systemctl enable conveyor-monitor
systemctl start conveyor-monitor
```

- You will need to update the demolauncher.service file with any changes to the MQTT broker. The environment vairable QUICKMQTT_URL should be updated to match your own config.

```
[Unit]
Description=Boot to Qt Demo Launcher
ConditionPathExists=!/usr/bin/b2qt
Conflicts=startupscreen.service
Wants=chromadb.service
After=chromadb.service network.target

[Service]
Type=idle
User=root
```

```

; StandardOutput=null
; StandardError=null
; LogLevelMax=error
; SyslogLevel=error
Environment="QUICKMQTT_URL=http://192.168.0.133:1883/"
Environment="QMONITOR_BACKEND_LIB_PATH=/var/QualcommProfile
r/libs/backends"
Environment="XDG_RUNTIME_DIR=/dev/socket/weston"
Environment="WAYLAND_DISPLAY=wayland-1"
Environment="QT_QPA_PLATFORM=wayland-egl"
Environment="PATH=/var/QualcommProfiler/bins:/usr/local/sbi
n:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
Environment="LD_LIBRARY_PATH=$LD_LIBRARY_PATH:/var/Qualcomm
Profiler/libs"
WorkingDirectory=/home/jari
ExecStartPre=/bin/sleep 1
ExecStart=-/usr/bin/sh -c "/home/jari/Responsive_ScadaApp"
Restart=always
RestartSec=1

[Install]
WantedBy=multi-user.target

```

- The output from the serial reader will be logged on the status and should look something like this:

```

Mar 26 13:58:18 qcs9075-iq-9075-evk python3[543269]: INFO m
qtt_publisher: Published EdgeImpulse/Workshop/Dashboard/Lin
elvoltage -> 0.0
Mar 26 13:58:18 qcs9075-iq-9075-evk python3[543269]: INFO m
qtt_publisher: Published EdgeImpulse/Workshop/Dashboard/Lin
elcurrent -> -0.0
Mar 26 13:58:18 qcs9075-iq-9075-evk python3[543269]: INFO m
qtt_publisher: Published EdgeImpulse/Workshop/Dashboard/Lin
elaccelX -> 0.018
Mar 26 13:58:18 qcs9075-iq-9075-evk python3[543269]: INFO m
qtt_publisher: Published EdgeImpulse/Workshop/Dashboard/Lin
elaccelY -> 0.992

```



```
Mar 26 13:58:18 qcs9075-iq-9075-evk python3[543269]: INFO m
qtt_publisher: Published EdgeImpulse/Workshop/Dashboard/Lin
elaccelZ -> 0.001
Mar 26 13:58:18 qcs9075-iq-9075-evk python3[543269]: INFO m
qtt_publisher: Published EdgeImpulse/Workshop/Dashboard/Lin
elconveyorStatus -> off
```

RTSP

This feature remains largely untested and should be checked it runs correctly on the hardware you have before using at a show. Performance has widely varied on different Hardware. If you do not need to send the camera stream across a routed network, I recommend hosting the RTSP server on the iQ9 itself.

RTSP Server

A RTSP Server can be launched using python. This was originally tested by running it locally on the iQ9 itself to serve a TIS Camera.

An example RTSP server in python can be found [here](#).

RTSP GStreamer Config

Below is example RTSP config for the TIS cameras. The IP Address will vary.

```
{
    "defect-detection-source-line1": "rtspsrc location=rtsp://127.0.0.1:8555/cam1 protocols=tcp latency=0 ! decodebin ! qtvtransform ! video/x-raw,format=RGBx,width=640,height=480 ! videoconvert ! video/x-raw,format=RGB ! edgeimpulsevidеоinfer ! edgeimpulseoverlay ! videoconvert",
    "defect-detection-source-line2": "rtspsrc location=rtsp://127.0.0.1:8556/cam2 protocols=tcp latency=0 ! decodebin ! qtvtransform ! video/x-raw,format=RGBx,width=640,height=480 ! videoconvert ! video/x-raw,format=RGB ! edgeimpulsevidеоinfer ! edgeimpulseoverlay ! videoconvert",
    "defect-detection-source-line3": "rtspsrc location=rtsp://127.0.0.1:8557/cam3 protocols=tcp latency=0 ! decodebin ! qtvtransform ! video/x-raw,format=RGBx,width=640,height=
```

```

480 ! videoconvert ! video/x-raw,format=RGB ! edgeimpulsevi
deoinfer ! edgeimpulseoverlay ! videoconvert"",
    "defect-detection-source-line4": "rtspsrc location=rts
p://127.0.0.1:8558/cam4 protocols=tcp latency=0 ! decodebin
! qtivtransform ! video/x-raw,format=RGBx,width=640,height=
480 ! videoconvert ! video/x-raw,format=RGB ! edgeimpulsevi
deoinfer ! edgeimpulseoverlay ! videoconvert"",
    "demo-mode-timer": 10,
    "restricted-area-path": [
        {
            "x": 180,
            "y": 260
        },
        {
            "x": 480,
            "y": 260
        },
        {
            "x": 480,
            "y": 480
        },
        {
            "x": 180,
            "y": 480
        }
    ]

```

Predictive Maintenance Sensor Fusion

For the predictive maintenance aspect of the demo a Arduino Nano BLE 33 board is used to monitor acceleration and power from the conveyor belt.

Data from the conveyor was collected to train a sensor fusion model which is able to classify the state of the belt. Four classes were created; Off, OK, slip and obstruction. The model collects data from the accelerometer and power sensor at a rate 100Hz for a 1 second window and classifies the state. An on board led will display the inference result. Blue for Off, Green for OK, Yellow for Slip and Red for obstruction.

The **Off** state is when the motor is powered off. The class was created so that the belt can sit idle without raising an alarm.

The **OK** state is characterised by the normal operation of the motor at various speeds. The model is able to generalise that the vibration pattern and the power profile of the motor is within an acceptable range.

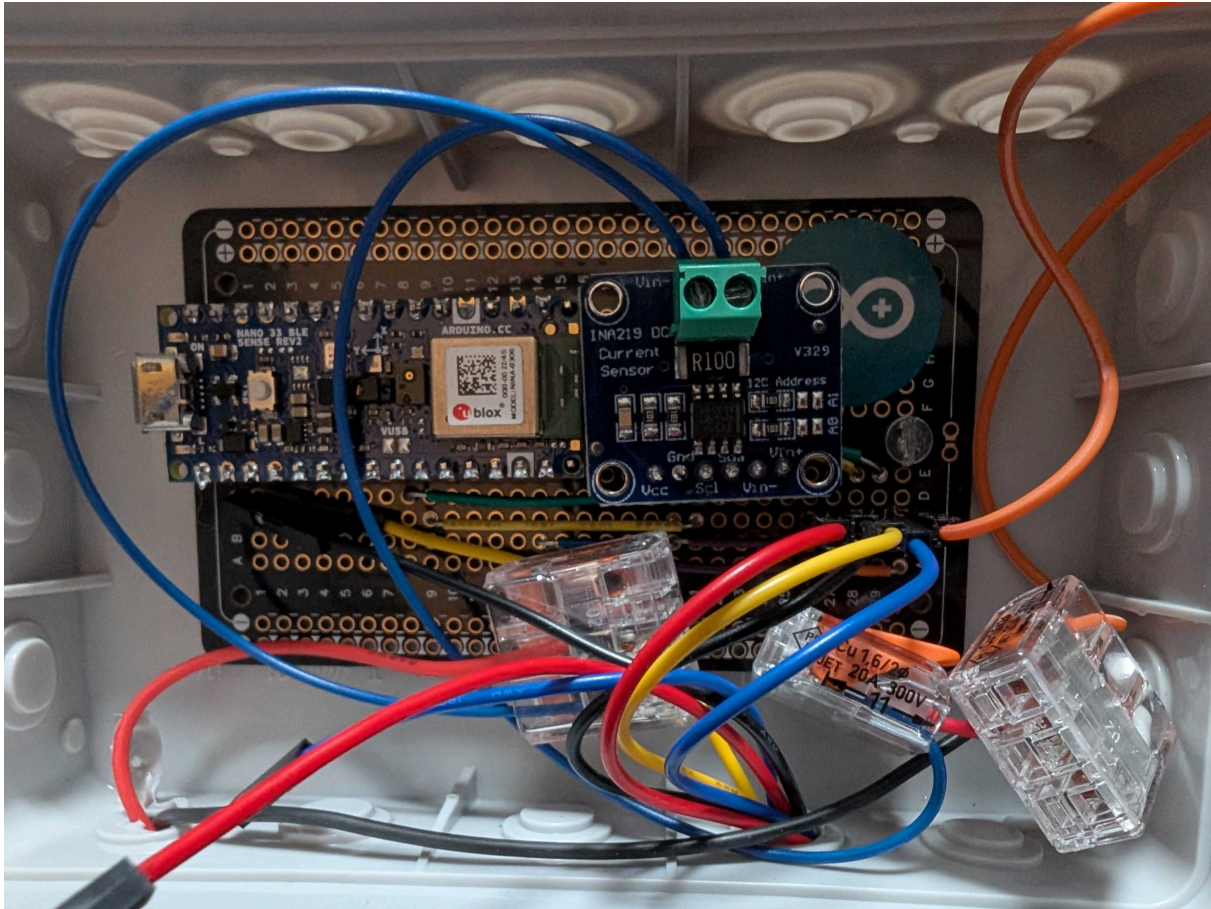
The **Slip** state is characterised by spikes in vibration amplitude. The data for this class was created by damaging the belt drive from the motor to the conveyor. Every third tooth in the belt was intentionally damaged to create a repeating slip pattern. This fault class demonstrates the ability of the model to use vibration data alone to recognise a fault behaviour.

The **Obstruction** state is characterised by an increased the friction on the belt drive. Increasing the friction of the belt forces the motor to draw a larger current at a given voltage setting. The data for this class was collected by firmly grabbing the belt by hand allowing the belt to still move but at a slower speed. This created a way to demonstrate the models ability to detect a fault behaviour with only the power data alone. It creates an easy replicable fault behaviour while using the demo.

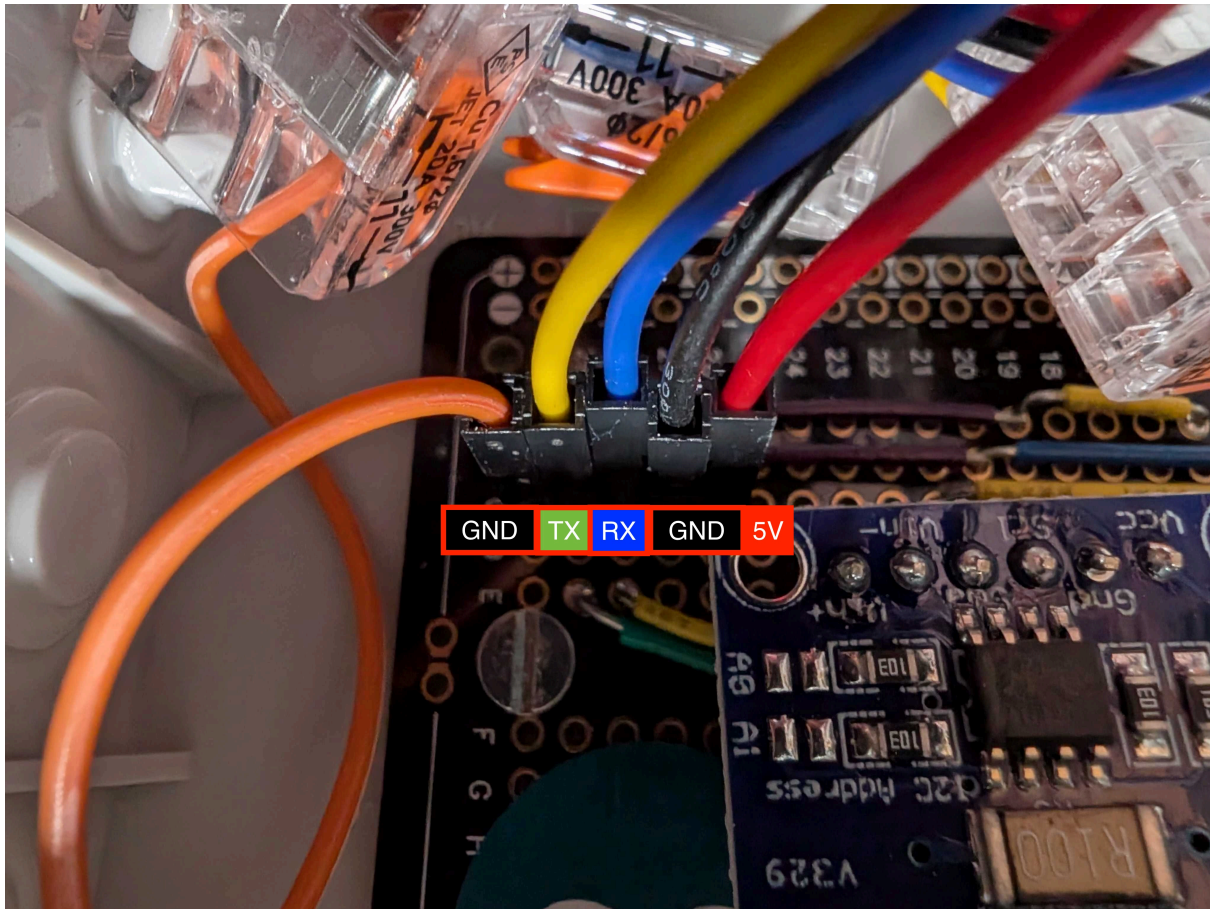
The Qt Responsive Scada App will read the inference result and will flash the corresponding line red if any result other than OK is received. The Arduino will also report RMS and mean values from the accelerometer and power sensor every second. These values will be plotted on a small dashboard within the app.

A bespoke PCB was made to be able to mount the Arduino Nano and the INA219 board. This made for an easy way to swap and change boards if needed.

The cable from the motor and power supply have been spliced to route the current through the INA219 sensor. Vin+ on the screw terminal of the INA219 is connect to the Positive of the Power Supply and Vin- is connected to the Positive terminal of the Motor. See the wiring diagram below for further details.



To connect the custom Bread Board to the USB to UART convertor follow the pin out below:



The Converter provides both power and comms to the Arduino nano. The reason a UART convertor is required is because of the noise the motor produces.

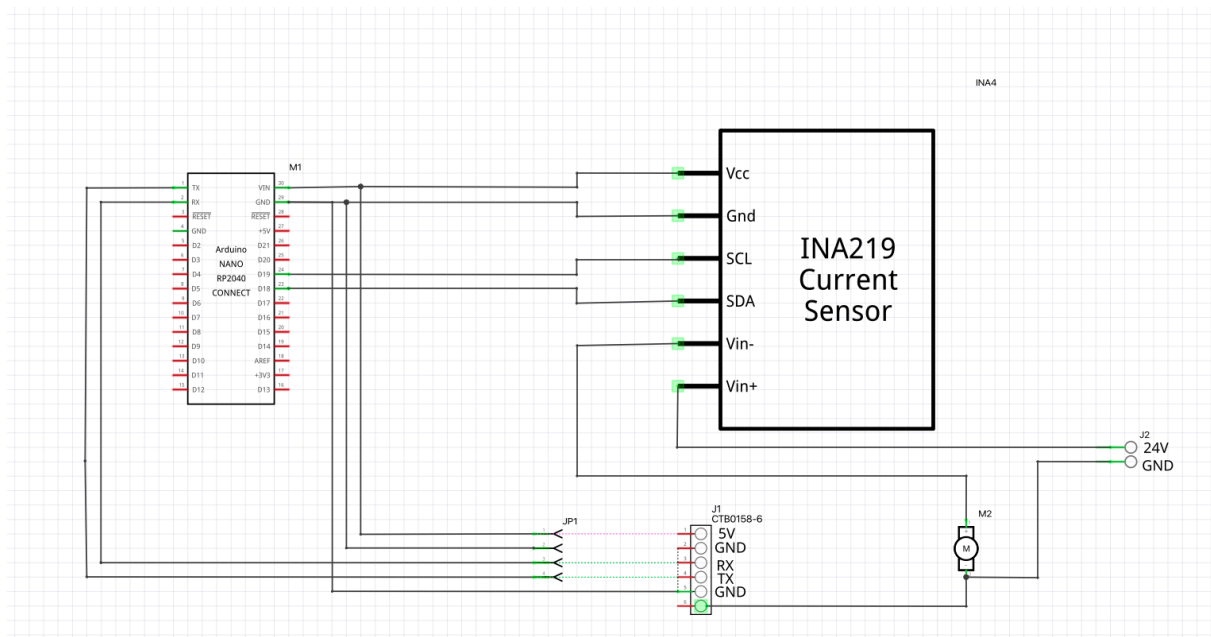
While it is possible to use the USB Serial output from the Arduino, the noise generated from the motor will cause most devices including the iQ9 to disconnect the Arduino as a USB device. The Converter decouples this noise and is much more stable.

Note: The fifth pin of the header is used to connect the motor ground to the INA219. A ground reference is need to accurately measure voltage. Without grounding the Voltage reading will be incorrect.

DO NOT USE THE MOTOR IN THE REVERSE DIRECTION, ONLY FORWARDS.

The motor should always be used in the forwards direction. Reversing the motor will also reverse the polarity to the INA219 and damage the board.

Here's the wiring diagram if you need to reproduce a similar bread board:



Using the Demo

On first detection for the Coffee Pods, put a defected pod in front of the camera. By default the first class detected will be assigned a red bounding box. The next object detected will be green. This will allow to defective parts to be bounded in red and good parts in green. This makes the visuals much clearer for the audience.

The video stream output can be selected to expand to the view from the TIS camera.

Click the demo radio button to launch a pre configured script. This will walk through several features of the Qt Responsive Scada app. Once finished remember to turn the button off.

Appendix

QT Creator

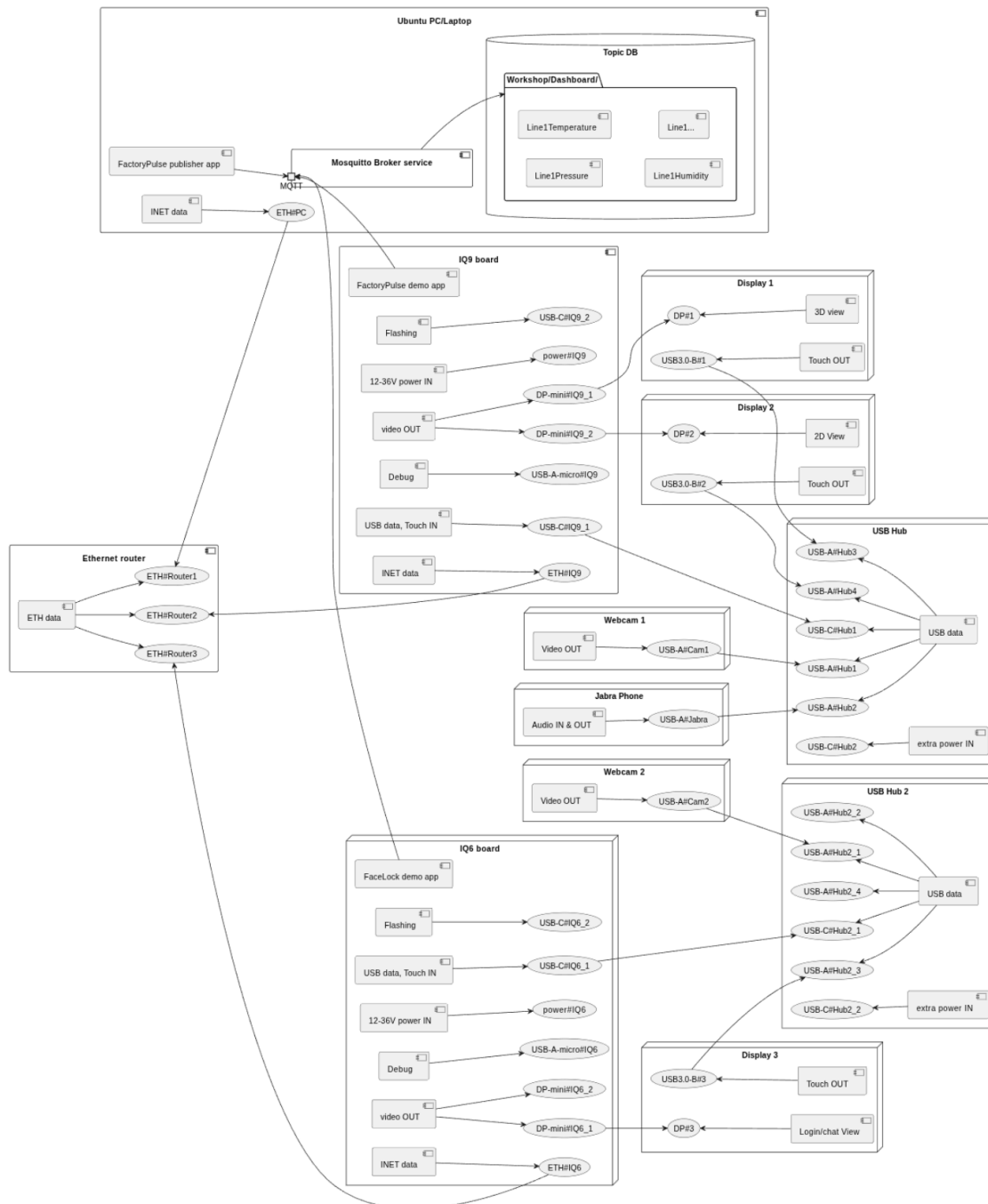
An active license is required for Qt Creator. This app is only really required if you wish to make changes to the Qt app itself. For example, we made some changes to the app for MWC to be able to launch on one screen.

Download Link

To ask about getting a temporary license, email Lazo Eric from the Qt Group: lazo.eric@qt.io

System Diagram for Embedded World 2026

This illustration show the a detailed system diagram for the demo used at embedded world 2026



Using a different Model

The defective pod model is included in the overlay shared in the file repository. It is possible to replace this model to update the demo.

Studio Project

You can copy the .so library from the [Google Drive](#) to

Trouble Shooting

- **No TIS Camera Output**

- First check the config.json is correct and the serial number used matches the camera in use
- View the log output of the the launcher service for further info
- Check the iQ9 can find the device on the network with tcam-ctrl. If nothing appear the camera is not on the same network.

```
tcam-ctrl -list
```

- Check that the tcam-capture app can view the camera stream. If you can't from this app it's likely a networking issue and you will need to reset the camera ip address as detailed above.
 - The Gstreamer pipeline can be replicated in a terminal shell, just export the same environment variables show above in the service file. If the Gstreamer pipeline works in a stand alone terminal, there's likely something wrong with the config. Check the journalctl output of the demo launcher service.
- **PPE Webcam Freeze** - This happened often after a long run of the app. Clear the Gstreamer cache and reboot to resolve.
 - **Blank Screen** - We have experience a loss of screen output when using the iQ9. Unfortunately the only way I've been able to recover from this is a full reboot.
 - **Screen Flicker** - We have experienced screen flicker also from the iQ9. I believe this can be attributed to long HDMI cables as I've only seen it when at trade shows and not at home. If possible reduce the run of cable.
 - **Non Responsive Application** - Check that the demo toggle button is turned off. Once the demo is complete the view will return to the main menu but the demo button will remain on and the other features will be disabled.