PRODUCT DOCUMENTATION

InSiSo Visual Detector

Version 4.7.21

docs.insisotech.com

InSiSo Technologies Private Limited
Registered office: WeWork Latitude, 10th Floor, RMZ Latitude Commercial, Hebbal Kempapura, Bangalore North, Bangalore 560024, Karnataka, India
CIN U62010KA2026PTC217872
Support ctrlxsupport@insisotech.com · contact@insisotech.com · insisotech.com

Contents

General

- How to read these manuals
- Support
- Disclaimer

Visual Detector

- Overview
- System requirements
- Installation
- Licensing
- Camera Connector
- Color Detection
- Object Detection
- Coordinate Mapping
- Detection History
- Diagnostics
- Data Layer
- Data Layer commands
- Captures for retraining
- Test setup
- FAQ
- Release notes

How to read these manuals

Convention	Meaning
Bold	A button, menu or field on screen, e.g. **Train Model**
Menu ▸ Item	A menu path, e.g. **Help ▸ Licence…**
`Monospace`	A file name, folder, command or Data Layer path
<kbd>B</kbd>	A keyboard shortcut

> **Note**
>
> Notes give extra information that helps you work faster.

> **Warning**
>
> Warnings describe something that can lose data or stop the machine.

> **Screenshot S-number — to come**
>
> Grey boxes mark screenshots still to be captured. The S-number matches the screenshot list.

Support

InSiSo Technologies Customer Support

Topic	Contact
ctrlX apps, licences, installation	ctrlxsupport@insisotech.com
Sales and general enquiries	contact@insisotech.com

When you report a problem, include:

- Product and version (**Help ▸ About** in Model Bench; the app version in ctrlX OS for Visual Detector)
- ctrlX device and ctrlX OS version
- Camera model
- What you did, what you expected, what happened
- Screenshots or log files if available
- For Visual Detector: the file from **Diagnostics ▸ Download diagnostics (.zip)**

Manufacturer

Company	InSiSo Technologies Private Limited
Registered office	WeWork Latitude, 10th Floor, RMZ Latitude Commercial, Hebbal Kempapura, Bangalore North, Bangalore 560024, Karnataka, India
Corporate Identity Number (CIN)	U62010KA2026PTC217872
Support	ctrlxsupport@insisotech.com
General enquiries	contact@insisotech.com
Website	insisotech.com

Disclaimer

Copyright © 2026 InSiSo Technologies Private Limited. All rights reserved.

- The image processing system and software described in these manuals are the intellectual property of InSiSo Technologies Private Limited, Bangalore, Karnataka, India.
- No part of this documentation may be reproduced without the written permission of the publisher.
- Technical changes reserved.
- Detections are not certified for personnel protection. Use rated safety devices for safety functions.

RUNTIME

InSiSo Visual Detector

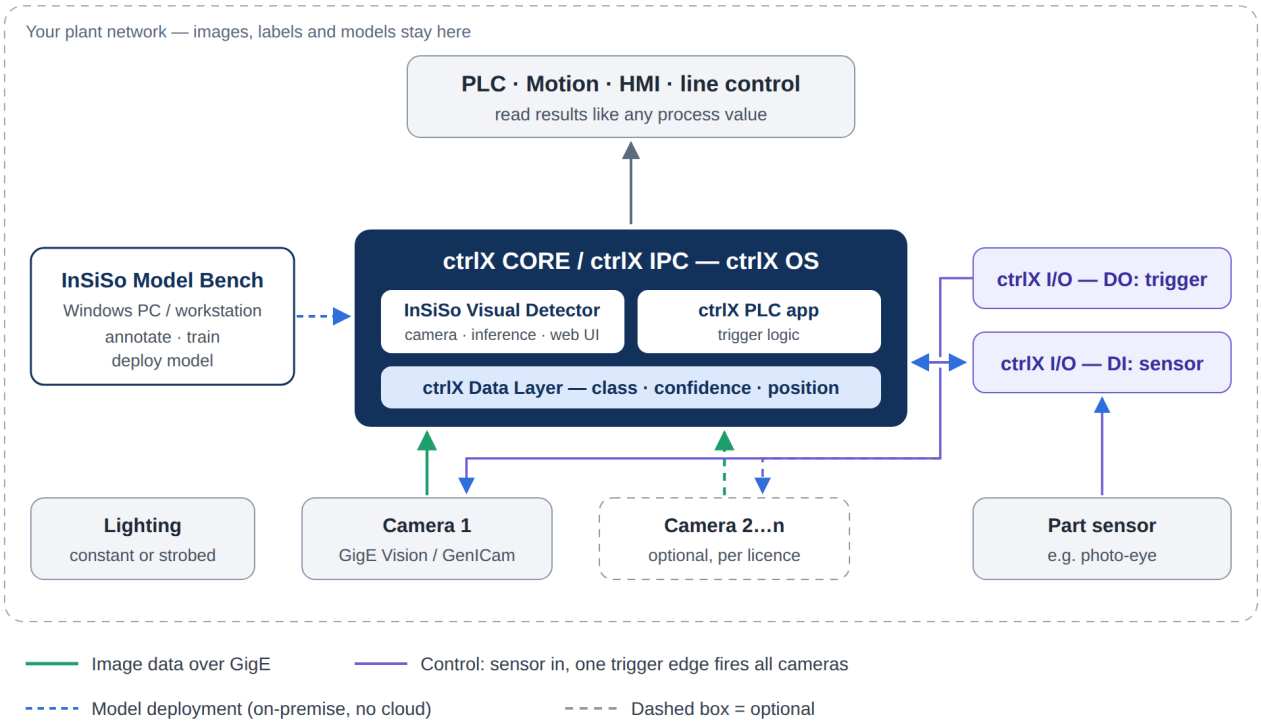
Finds parts by colour or with a model trained in InSiSo Model Bench, on live cameras, and publishes what it finds to the ctrlX Data Layer. It runs on ctrlX OS, on the ctrlX CORE or a ctrlX IPC.

Run it

1. Install the app and its licences on the ctrlX device.
2. Connect a camera on the **Camera Connector** page.
3. Tick the colours to find on Color Detection, or load a model on Object Detection.
4. Optional: calibrate Coordinate Mapping to get positions in machine millimetres.
5. From the PLC: wait for `ready`, write `trigger`, read the answer. See Data Layer.

After commissioning nobody needs to press anything. After every reboot the app comes back with its settings, connects its cameras and detects.

InSiSo Visual Detector on ctrlX OS — system architecture



What it does

1 **Detects** Coloured parts by colour range, and objects of the classes you trained, with a confidence per object.

2 **Locates** Each object's centre, size and orientation in pixels, and in machine millimetres with Coordinate Mapping.

3 **Publishes** Every picture's results to one Data Layer folder, as arrays, with a picture counter on every value.

4 **Records** A detection history with CSV export, and saved pictures for retraining in Model Bench.

The functions

Each function is a page in the app and has its own licence. The **Runtime** licence is needed for all of them.

Function	What it gives	Page
Camera Connector	GigE Vision, USB and RTSP/HTTP cameras; free run, software or hardware trigger	Camera Connector
Color Detection	Every blob of a ticked colour, with its centre and size in pixels	Color Detection
Object Detection	Every object a Model Bench model finds, with class and confidence	Object Detection
Code Reading	The Data Matrix text inside each detected label	Object Detection
3D Dimensioning	Length, width and height of each detected parcel in millimetres	Object Detection
Coordinate Mapping	Each object's position in machine millimetres	Coordinate Mapping
Detection History	A record of what was found, kept across power cycles	Detection History
Diagnostics	Is it ready, a trigger test, the licence, performance, support files	Diagnostics

The app's **Help** page lists every Data Layer node and command of the installed version.

In this manual

> **About the screenshots**
>
> The screenshots show the app's pages with a simulated camera: a carton with a label, and red, orange and blue parts on a belt. The numbers in them (frame rates, timings, positions) come from that demonstration setup, not from a measurement on a ctrlX device.

System requirements

Item	Specification
Platform	ctrlX OS app (a snap), installed with the ctrlX OS App Manager
ctrlX OS	4.x. Tested on ctrlX OS 4.6
Devices	ctrlX CORE X3 (arm64); ctrlX CORE X5, ctrlX CORE X7, ctrlX IPC and ctrlX CORE virtual (amd64)
Cameras	GenICam / GigE Vision industrial cameras, USB / UVC cameras, RTSP and HTTP streams
Cameras per device	One **Camera Connector** licence per connected camera
Models	`.onnx` exported from InSiSo Model Bench; for the Hailo-8 module, a `.zip` with the `.hef`, `metadata.json` and `dgqp.npz`
Accelerators (optional)	Hailo-8 M.2 module on a ctrlX CORE; NVIDIA GPU on a ctrlX IPC with the NVIDIA extension
Browser	Any current browser, through the ctrlX OS web interface
Cloud connection	Not required. Pictures, models and results stay on the device

Which snap

Pick the snap by the device's processor, not by file size. A snap installs only on the architecture it was built for.

Snap	Devices	Carries
`ctrlx-insiso-vision-detector_<version>_arm64.snap`	ctrlX CORE X3	Hailo-8 support
`ctrlx-insiso-vision-detector_<version>_amd64.snap`	ctrlX CORE X5, ctrlX CORE X7, ctrlX IPC, ctrlX CORE virtual	Hailo-8 support and the NVIDIA GPU runtime, about 3.2 GB

On a device without an NVIDIA GPU, the amd64 snap's GPU runtime is not used.

Version numbers

The first number of the version is the ctrlX OS major version the app is built for: **4**.x.x runs on ctrlX OS 4.x. The other two numbers count releases, for example 4.7.22 for a fix and 4.8.0 for a new function.

Test builds before 4.7.18 had higher numbers, up to 7.2.17. They are all older than 4.7.18.

Accelerators

Hailo-8. A `.hef` model runs on the Hailo-8 M.2 module. The Hailo runtime (HailoRT) must be the same version as the module's driver, which comes with ctrlX OS. The snaps carry HailoRT 4.23.0; the arm64

snap also carries 4.24.0. On a device with another driver version, `.hef` models do not load, and **Diagnostics** shows the driver version.

NVIDIA GPU. On a ctrlX IPC with the NVIDIA extension (NVIDIA RTX 2000 Ada Generation), an `.onnx` model runs on the GPU with the same file. The amd64 snap carries NVIDIA's driver files for every NVIDIA driver from 535 to 580 that Ubuntu 24.04 has shipped. It uses the one that matches the device's driver. On a driver version it does not carry, the model runs on the CPU and **Diagnostics** shows the version.

Without an accelerator, `.onnx` models run on the CPU.

Installation

Install the app

1. Open the ctrlX OS web interface of the device and sign in.
2. Switch the device to **Service** mode. Apps can only be installed in Service mode.
3. Go to **Settings ▸ Apps & Licenses**.
4. Install **InSiSo Visual Detector** from the ctrlX Store, or choose **Install from file** and pick the `.snap` for the device's architecture (see Which snap). A file that does not come from the ctrlX Store installs only when the device allows installation from an unknown source.
5. Switch the device back to **Operating** mode.

The app starts by itself. **InSiSo Visual Detector** appears in the ctrlX OS sidebar.

Sign in

The app uses your ctrlX OS user. There is no second account and no login form. Open a page of **InSiSo Visual Detector** from the ctrlX OS sidebar while you are signed in to ctrlX OS: the sidebar link carries your ctrlX OS sign-in to the app, and the app's own links keep it for twelve hours. Every sidebar click starts them again.

If ctrlX OS cannot vouch for you, the app shows its own login and says why the ctrlX OS sign-in did not work. Open the page from the ctrlX OS sidebar, not from a bookmark or a typed address. If the login still appears, log out of ctrlX OS, log in again, and open the page from the sidebar.

Activate the licences

1. Buy the licences in the ctrlX Store, or start a trial.
2. Install them on the device in **Settings ▸ Apps & Licenses**.
3. Open **InSiSo Visual Detector ▸ Diagnostics**. The **Licence** card lists every function as **unlocked** or names the licence it needs.

The app takes its licence seats by itself at start-up, and retries until the licence service answers. See Licensing for what each licence unlocks.

Interfaces

The app reaches the ctrlX Data Layer, the licence service, cameras and accelerators through snap interfaces. Most connect by themselves. If one is not connected, the app says so: **Diagnostics** names the Data Layer and the licence service, and the GPU and Hailo cards name their own.

On a device where you have a shell, connect a missing interface and restart the app:

```
sudo snap connect ctrlx-insiso-vision-detector:licensing-service
sudo snap connect ctrlx-insiso-vision-detector:datalayer rexroth-
automationcore:datalayer
sudo snap restart ctrlx-insiso-vision-detector
```

Interface	Needed for
`licensing-service`	Every licensed function
`datalayer`	Publishing to the ctrlX Data Layer, and Data Layer commands
`camera`	USB / UVC cameras
`hailo-0`	A `.hef` model on the Hailo-8 module: `sudo snap connect ctrlx-insiso-vision-detector:hailo-0 rexroth-arch02-hw:hailo-0`
`removable-media`	Detection History, saved pictures or recordings on a USB drive
`opengl`, `gpu-2404`	The NVIDIA GPU on a ctrlX IPC

`network` and `network-bind` connect by themselves. GigE Vision cameras need nothing extra.

Update

1. Install the new `.snap` over the old one, as above.
2. **Reboot the device once.** This frees any licence seats the previous version still held.

Models, settings, colours, calibration, saved pictures and the detection history are kept. After the reboot the app connects its cameras and detects with no button pressed.

Uninstall

Remove the app in **Settings ▸ Apps & Licenses**. Download what you want to keep first: the history from **Detection History**, saved pictures from **Camera Connector**, and the diagnostics from **Diagnostics**.

Licensing

InSiSo Visual Detector is licensed per function through the ctrlX License Manager. Every licence can be started as a trial. The app installs and runs without a licence; a function without its licence stays locked and names the licence it needs.

The licences

Licence	Unlocks
Runtime	The app itself and the ctrlX Data Layer. **Required**: every other function needs it
1 Camera Connector AddOn	One connected camera. Install one per camera; they add up
Color Detection	Color Detection
Object Detection AddOn	Object Detection with a Model Bench model
Machine Coordinate Transformation AddOn	Coordinate Mapping: positions in machine millimetres
Code Reading AddOn	Code Reading: the Data Matrix text inside detected labels
3D Dimensioning AddOn	3D Dimensioning: parcel length, width and height
Parcel Tracking AddOn	The parcel cell functions on the Object Detection page. Not described in this manual
Vision Guidance Package	Runtime, Camera Connector, Object Detection and Machine Coordinate Transformation, as one licence
Parcel Tracking Package	Runtime, Camera Connector, Object Detection, Code Reading, 3D Dimensioning and Parcel Tracking, as one licence

Detection History, **Diagnostics** and **Help** need no licence of their own.

Seats

A ctrlX licence is a pool of seats. The app holds one seat of each licence it uses while it runs:

- It asks for a package first, so one seat covers everything the package includes.
- It takes a single licence only for what no package covers.
- Each connected camera takes one **Camera Connector** seat. Two cameras need two.
- Seats are taken at start-up and given back when the app stops. After an update, reboot the device once to free seats the previous version held.

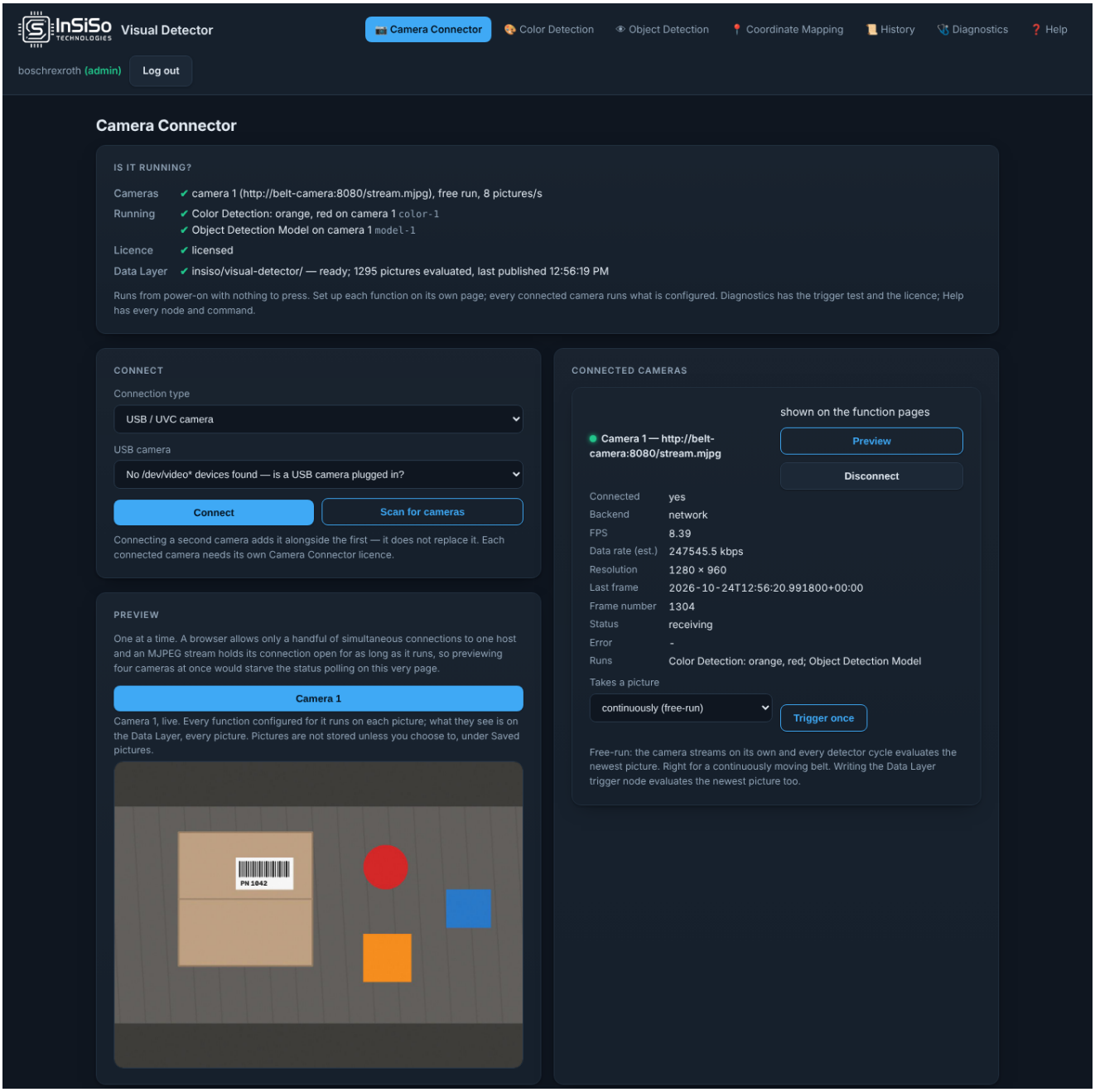
See what is licensed

Open **Diagnostics**. The **Licence** card shows the ctrlX licence state and every function as **unlocked**, **included** or locked. **Re-check licence** asks the License Manager again. **Run licensing diagnosis** shows each licence, whether the device holds it, and why a seat could not be taken.

A Data Layer command for a function that is not licensed answers in `command-result` with `"licensed": false` and the licence it needs.

Camera Connector

The **Camera Connector** page is the app's first page. It says whether the app is running, connects cameras and sets when each camera takes a picture.



Is it running?

The card at the top answers in one line each:

Line	Says
Cameras	Each camera, its trigger mode and pictures per second
Running	Each detector and its camera, for example *Color Detection: orange, red on camera 1*

Line	Says
Licence	Whether the Runtime licence is held
Data Layer	`insiso/visual-detector/`, whether it is ready, and the pictures evaluated

A ✘ comes with the reason and what to do.

Connect a camera

1. Under **Connect**, choose the **Connection type**: **GenICam / GigE Vision**, **USB / UVC camera**, **RTSP stream** or **HTTP stream**.
2. For a GigE Vision camera, press **Scan for cameras** and pick it. For a stream, type its address.
3. Press **Connect**.

The camera appears under **Connected cameras** with its state, resolution and frame rate. A color detector starts on it at once, and the loaded model runs on it too. Connecting a second camera adds it beside the first. Each connected camera needs its own Camera Connector licence.

After a reboot or a camera power cycle the app reconnects the camera by itself, with its trigger mode.

When the camera takes a picture

Set **Takes a picture** for each camera:

Setting	Use
continuously (free-run)	The camera streams; every picture is evaluated. For a moving belt and for commissioning
when triggered (software)	One picture each time the PLC writes `trigger`, or you press **Trigger once**
on its trigger line (hardware)	One picture per pulse on the camera's trigger input. GenICam cameras only

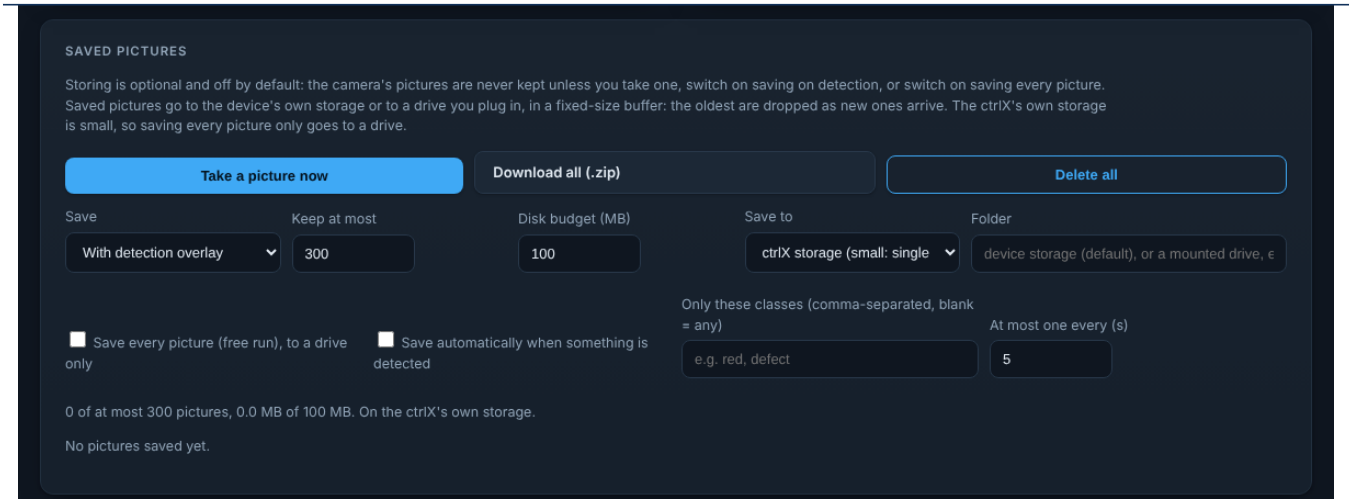
A software trigger fires every connected camera at the same moment. The answer appears on the Data Layer; see Data Layer.

Preview

Preview shows one camera's live picture. Use the camera buttons to switch. Only one preview runs at a time, so the page stays fast.

Saved pictures

Pictures are not stored unless you ask for it.



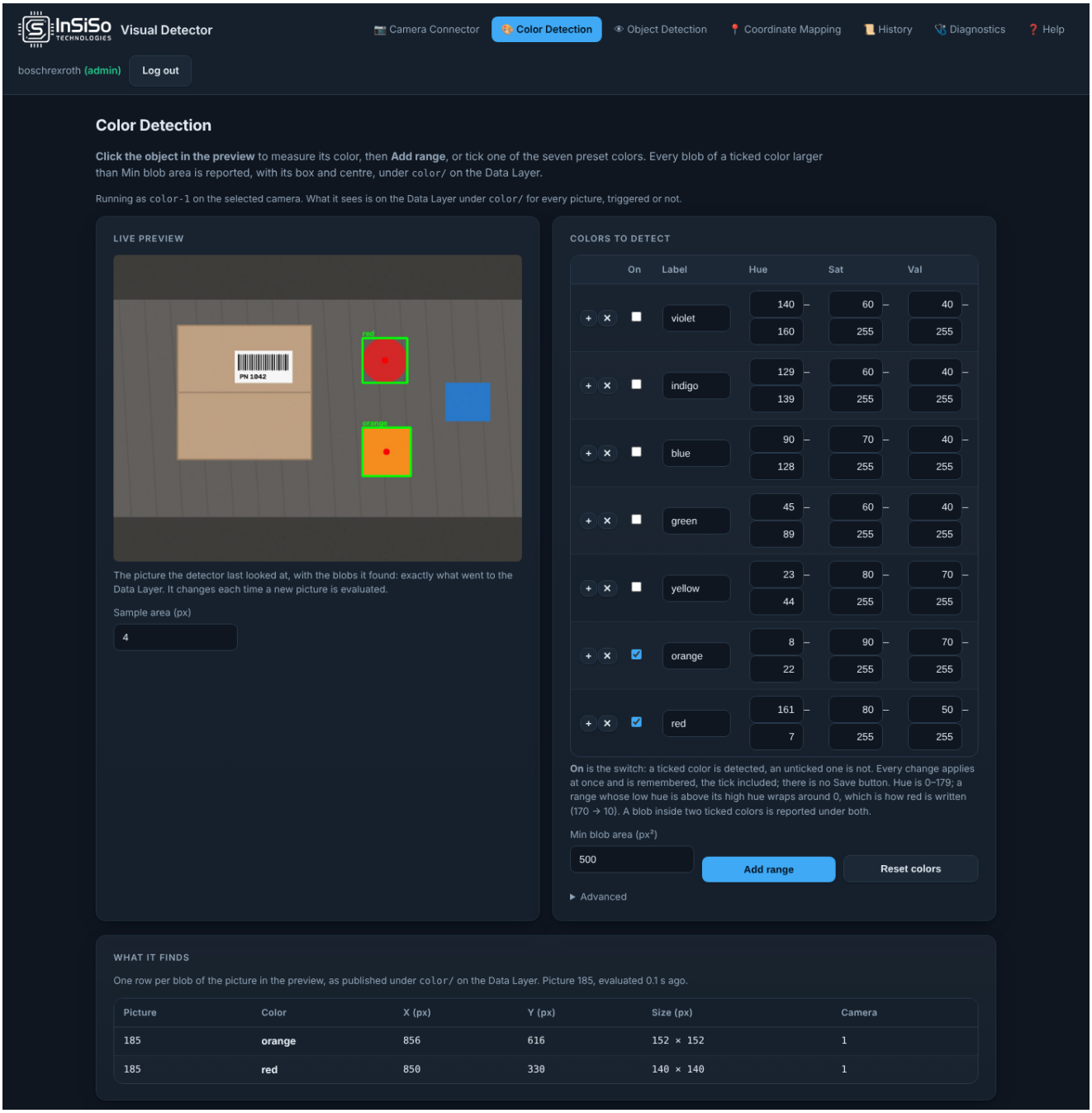
- **Take a picture now** saves one picture.
- **Save automatically when something is detected** saves a picture when a detector finds something, at most one every few seconds (**At most one every (s)**). **Only these classes** limits it to some classes.
- **Save every picture (free run), to a drive only** keeps every picture, on a USB drive.
- **Save** chooses the detection overlay, the raw camera picture (for retraining), or both.
- **Keep at most** and **Disk budget (MB)** cap the buffer. The oldest pictures go first.
- **Download all (.zip)** downloads every saved picture with its detections. See Captures for retraining.

HMI stream link

HMI stream link gives a video-only page of a camera, for a ctrlX HMI web-view widget. Opening it never slows the detection.

Color Detection

Color Detection finds parts by colour, with no model to train. Every blob of a ticked colour larger than **Min blob area** is reported, with its centre and size.



Set it up

1. Connect a camera on Camera Connector. Its color detector, `color-<camera>`, starts by itself.
2. Tick **On** for the colours to find. Seven preset colours are listed, all off: violet, indigo, blue, green, yellow, orange and red. Nothing is detected until a colour is ticked.
3. Or click the part in **Live preview** to measure its colour, then press **Add range**. **Sample area (px)** is the size of the spot measured.

4. Set **Min blob area (px²)**. Smaller blobs are ignored. It applies to every colour.

Every change applies at once and is remembered. There is no Save button. **Reset colors** goes back to the seven presets, all off.

Colour ranges

Each colour is a range of **Hue** (0–179), **Sat** and **Val** (0–255). A range whose low hue is above its high hue wraps around 0: that is how red is written, for example 161 to 7.

To fine-tune a colour, click the part in **Live preview**, then press **+** on that colour's row: the range widens to include the sampled pixel. **×** deletes a range.

Every ticked colour is looked for on its own. A blob that fits two ticked colours is reported under both. The order of the list is only the order it is shown in.

Advanced holds **Median blur (0 = off)** and **Max detections**, the most blobs reported per picture.

What it shows

Live preview is the picture the detector last looked at, with the blobs it found: exactly what went to the Data Layer. The page runs no detection of its own.

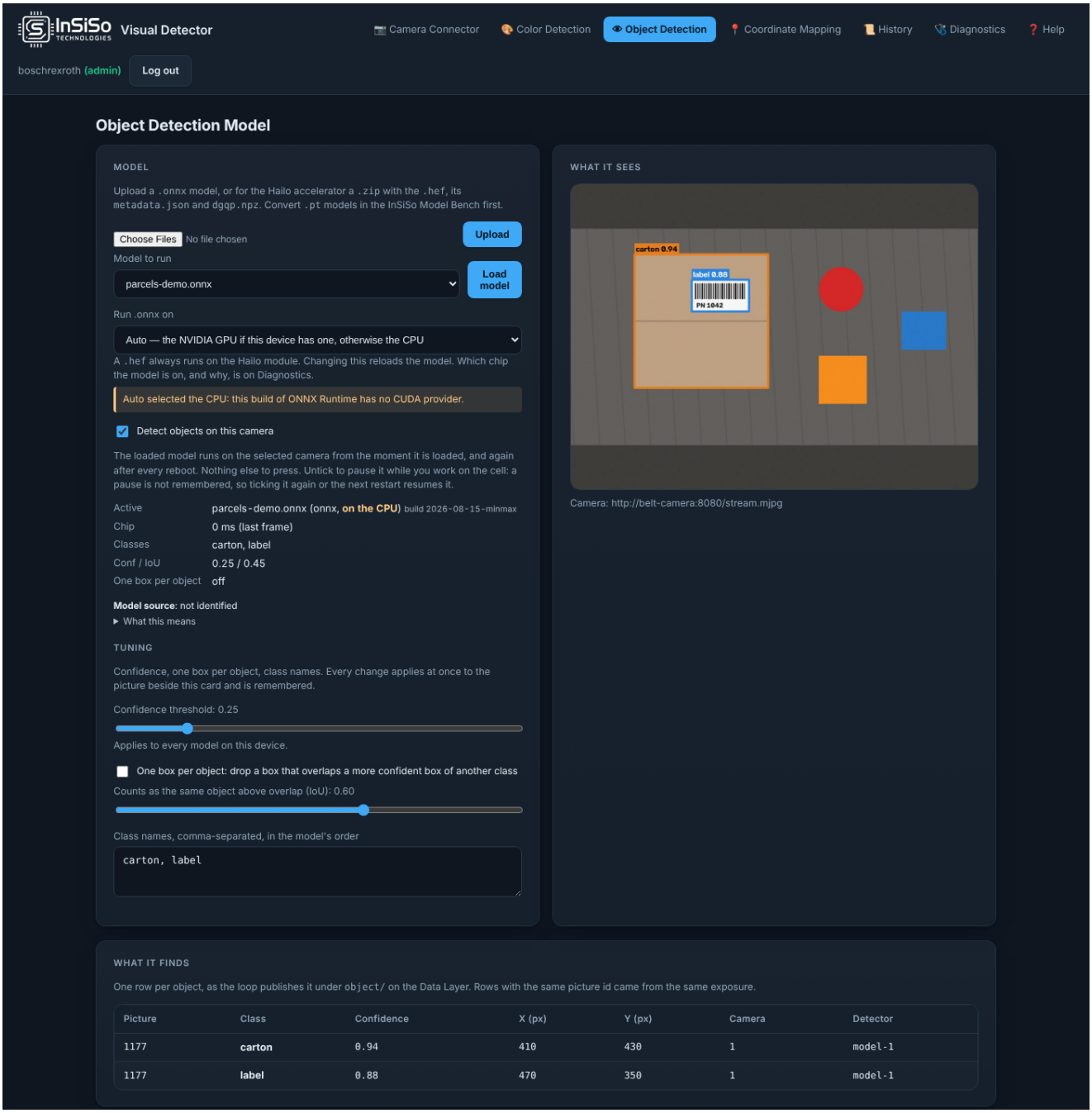
What it finds lists one row per blob: the picture number, the colour, the centre (**X (px)**, **Y (px)**), the size and the camera. The same values are on the Data Layer under `color/`.

Colour positions and sizes are in camera pixels. The picture is evaluated on a copy at most 1280 px wide, and the results are scaled back to the camera's pixels, to within about 2 px on a 2500 px camera.

Each camera has its own color detector and its own colours. A second camera's detector starts as a copy of the first one's.

Object Detection

Object Detection runs a model trained in InSiSo Model Bench on the camera's pictures. It reports every object of the trained classes with its class, confidence, centre and size.



Load a model

1. Export the model from Model Bench: an `.onnx` file, or for the Hailo-8 module a `.zip` with the `.hef`, `metadata.json` and `dgqp.npz`. Convert `.pt` models in Model Bench first.
2. Under **Model**, choose the file and press **Upload**.
3. Pick it in **Model to run** and press **Load model**.

The model runs on the selected camera from the moment it is loaded, and again after every reboot. Nothing else needs to be pressed. Class names, and the class colours from Model Bench's Manage Classes, travel inside the file. The boxes are drawn in those colours.

Where it runs

Run .onnx on chooses the processor for an `.onnx` model. The choice is remembered.

Choice	Runs on
Auto — the NVIDIA GPU if this device has one, otherwise the CPU	The default
NVIDIA GPU (CUDA)	The NVIDIA GPU; the CPU, with the reason, if it cannot start
NVIDIA GPU (TensorRT …)	In 4.7.21 TensorRT is not included, so this runs on CUDA
CPU only	The CPU, for example to leave the GPU to another app

A `.hef` always runs on the Hailo-8 module. The line under the choice says where the model actually runs and why. **Diagnostics ▸ Accelerators** has the details.

Pause and resume

Detect objects on this camera switches the model on and off for the camera. Untick it to pause while you work on the cell. A pause is not remembered: ticking it again, or the next restart, resumes detection.

Tuning

Every change applies at once to the picture beside the card and is remembered.

- **Confidence threshold**: objects below it are not reported. It applies to every model on the device.
- **One box per object**: drops a box that overlaps a more confident box of another class by more than **Counts as the same object above overlap (IoU)**.
- **Class names, comma-separated, in the model's order**: renames the classes, for example when the model carries none.

What it shows

What it sees is the camera's latest picture with the model's boxes. **What it finds** lists one row per object: picture number, class, confidence, centre (**X (px)**, **Y (px)**), camera and detector. The same values are on the Data Layer under `object/`, and in machine millimetres under `coordinates/` once Coordinate Mapping is calibrated.

Models give upright boxes, so `orientation-deg` is 0 and `orientation-valid` is FALSE unless the model predicts corners.

Commissioning tool: run the model on a video file runs the loaded model on an uploaded video, to check a model away from the line.

A camera can run more than one model. Each runs as its own detector (`model-1`, `model-2`, …). Add one with the Data Layer command `create-detector`; see Data Layer commands.

Code Reading

With the **Code Reading** licence, the app reads the Data Matrix inside each detected box, for example on a parcel label. It reads on the camera's full-resolution picture, after the confidence threshold, so only reported objects are read.

Switch it on per detector with the Data Layer command:

```
configure-detector {"id": "model-1", "read-codes": true}
```

`code-depth` chooses `deep` (the default, slower and more thorough) or `fast`. The text read is on the Data Layer in `object/code[]`, at the same index as its object.

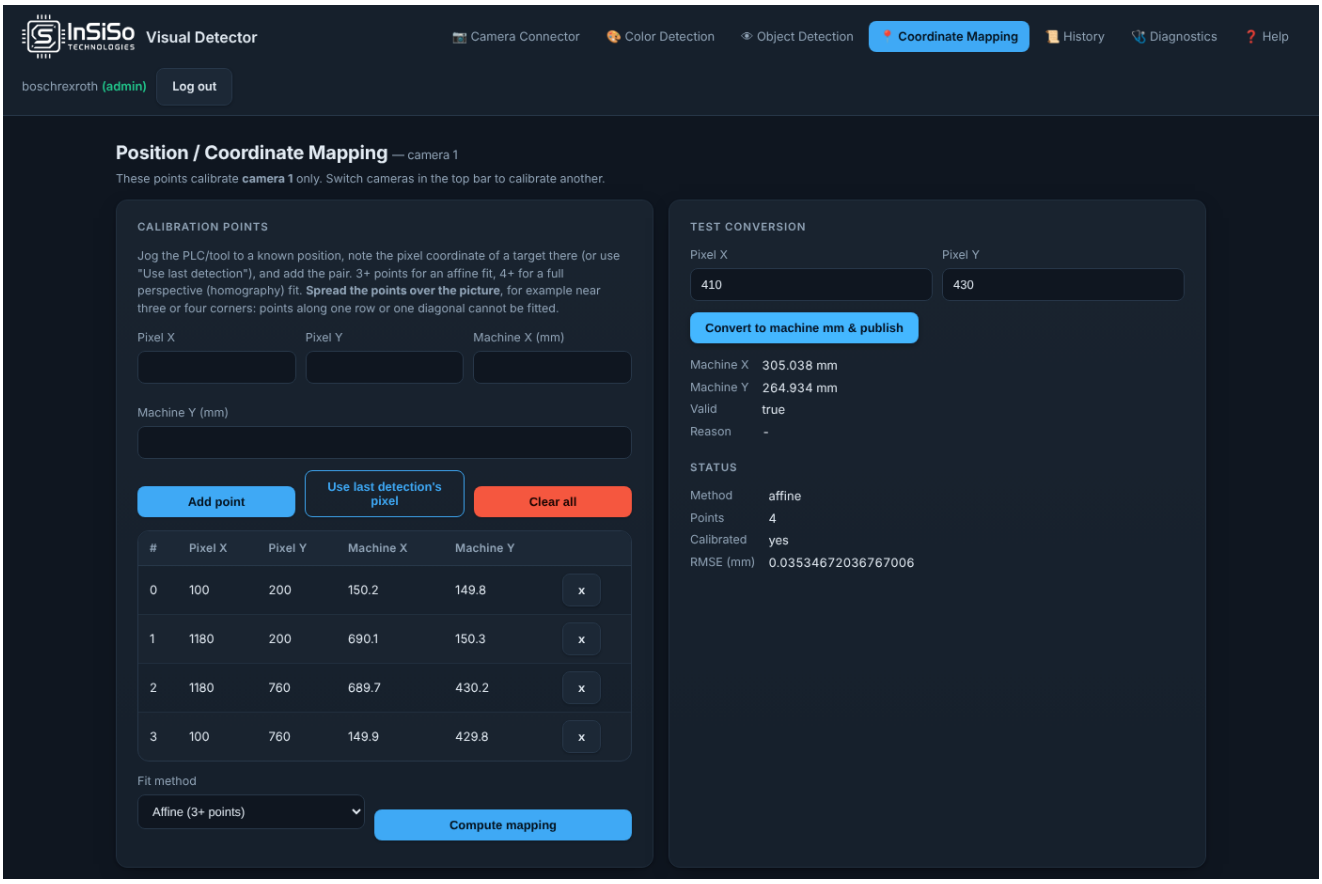
3D Dimensioning

With the **3D Dimensioning** licence, the app gives each detected parcel's length and width in millimetres, from the four corners of its top face. This needs a model trained in Model Bench with **Predict package corners**, the camera's mounting geometry, and the parcel's height. The results are on the Data Layer in `coordinates/length-mm`, `width-mm` and `height-mm`.

> **Draft**
>
> In 4.7.21 the parcel height is a fixed assumed height unless a measured height is supplied, and `json` says which (`height_measured`). The camera geometry is set up with InSiSo during commissioning. No accuracy figure is given: measure parcels against a tape on your cell.

Coordinate Mapping

Coordinate Mapping turns a position in the camera picture (pixels) into a position on the machine (millimetres), so a PLC or a robot can use it directly. It needs the **Machine Coordinate Transformation** licence.



Calibrate

Each camera is calibrated on its own. The page title names the camera; switch cameras in the top bar.

1. Put a target where the camera sees it and jog the PLC or tool to it. Note the machine position.
2. Enter the target's **Pixel X** and **Pixel Y**, or press **Use last detection's pixel** to take the centre of the last object found.
3. Enter **Machine X (mm)** and **Machine Y (mm)**, and press **Add point**.
4. Repeat at three or four places spread over the picture, for example near the corners. Points along one row or one diagonal cannot be fitted.
5. Choose the **Fit method**: **Affine (3+ points)**, or **Homography / perspective (4+ points)** when the camera looks at the belt at an angle. Press **Compute mapping**.

Status shows the method, the number of points, whether the camera is calibrated, and the fit's error (**RMSE (mm)**). **Clear all** removes the points.

Check it

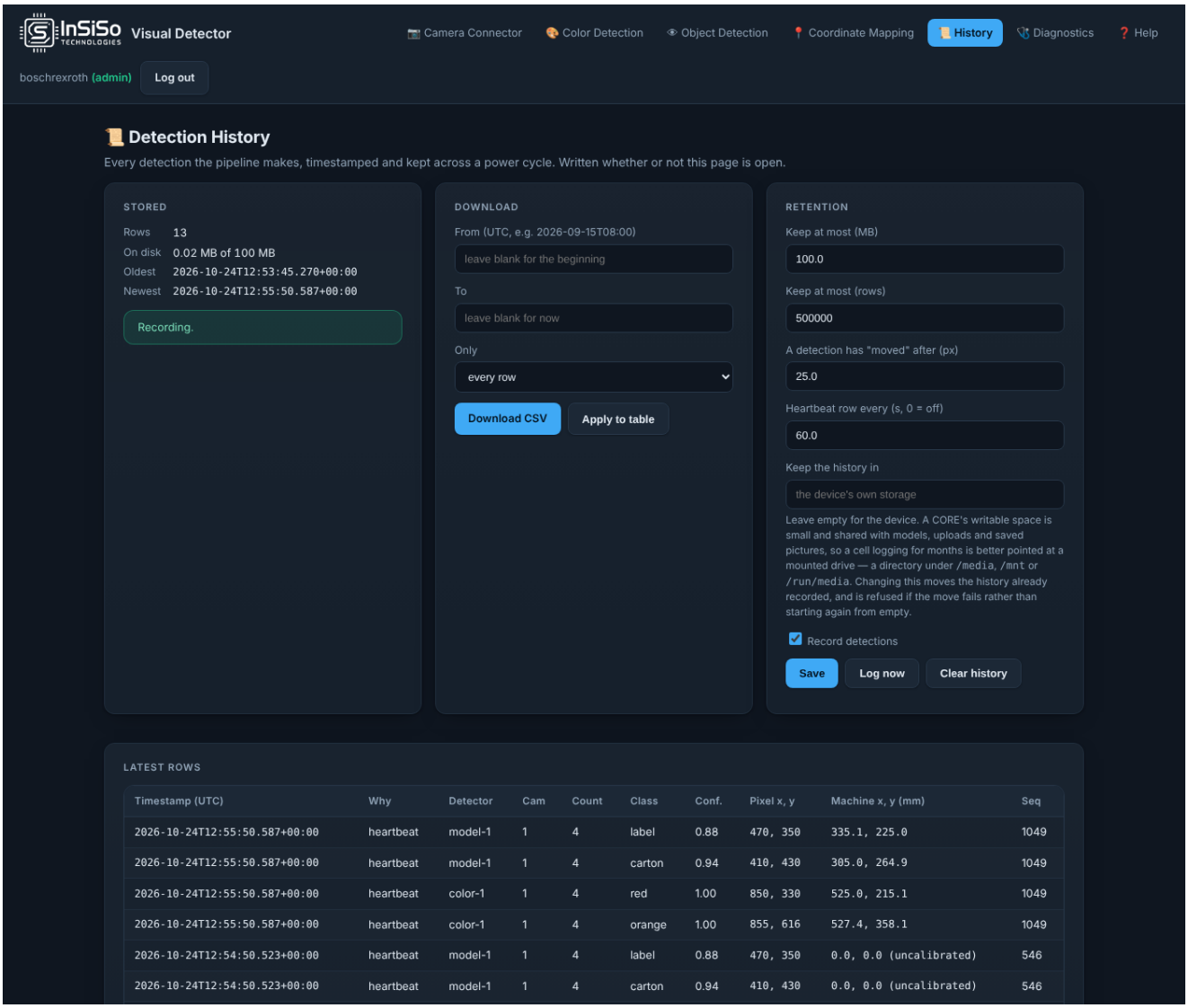
Under **Test conversion**, enter a pixel position and press **Convert to machine mm & publish**. The page shows the machine position and whether it is valid.

What it publishes

Once calibrated, every object from Object Detection is also published in machine millimetres under `coordinates/` on the Data Layer: `x-mm`, `y-mm` and `position-valid`. Element *i* of `coordinates/` is element *i* of `object/`. Colour blobs stay in pixels.

Detection History

Detection History is a timestamped record of what the app detected. It is written whether or not the page is open, and kept across power cycles. The Data Layer holds the latest values; the history is the series behind them.



What is recorded

A row is written when something happened, not for every picture:

- the class, the count or the position changed (**A detection has "moved" after (px)** sets how far counts as a move);
- a trigger was evaluated;
- a heartbeat interval passed with nothing going on (**Heartbeat row every (s, 0 = off)**).

Each row has the time (UTC), why it was written, the detector, camera, count, class, confidence, pixel position and machine position.

Record detections switches recording on and off. **Log now** writes a row at once.

Download

1. Under **Download**, enter **From** and **To** in UTC, or leave them blank for everything.
2. Choose **Only**: **every row**, **changes**, **triggers** or **heartbeats**.
3. Press **Download CSV**. **Apply to table** shows the same selection under **Latest rows**.

Retention

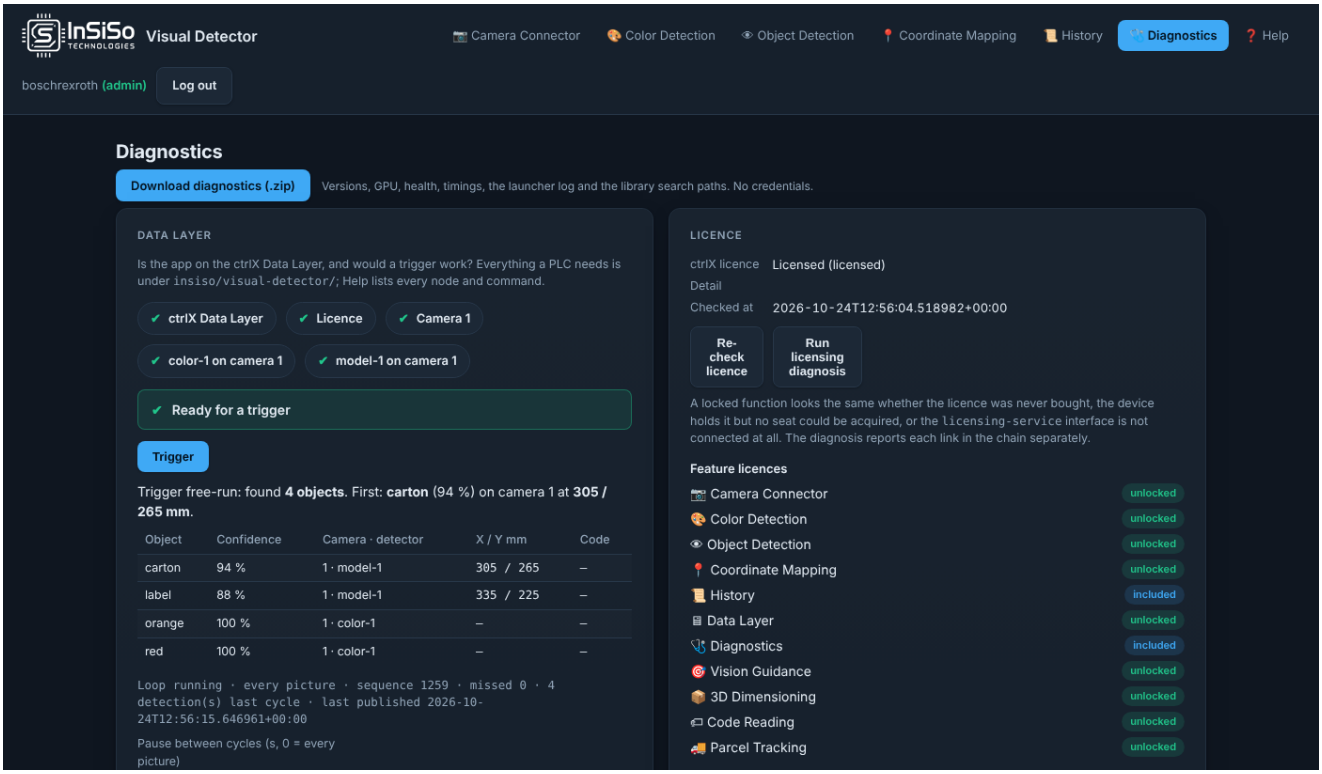
The history is capped so it cannot fill the device: **Keep at most (MB)** and **Keep at most (rows)**. The oldest rows go first.

Keep the history in is where the history is kept. Leave it empty for the device's own storage. A ctrlX CORE has little writable space, so for months of logging point it at a USB drive: a directory under `/media`, `/mnt` or `/run/media`. Changing it moves the existing history with it.

Press **Save** after changing the retention settings. **Clear history** deletes every row and cannot be undone: download the CSV first.

Diagnostics

Diagnostics answers "would a trigger work, and if not, why?" It also shows the licence, the performance per picture, the accelerators and the storage, and gives the file to send to support.



Data Layer

The chips at the top answer one question each: **ctrlX Data Layer**, **Licence**, each camera and each detector. A ✘ says why and what to do. **Ready for a trigger** means a PLC's trigger will work now.

Trigger writes the Data Layer's `trigger` exactly as a PLC would, and shows the answer: every object found, its confidence, its camera and detector, its machine position and any code read. If this button works, a PLC can do the same.

Pause between cycles (s, 0 = every picture) slows the free-run loop, for example to save CPU. A PLC can stop the loop with the command `stop-detection`; a reboot always starts it again.

Licence

ctrlX licence shows the platform licence state. **Feature licences** lists every function as **unlocked**, **included** or locked. **Re-check licence** asks the License Manager again. **Run licensing diagnosis** shows each licence, whether the device holds it, and why a seat could not be taken. See Licensing.

Performance

Performance shows how long each picture takes, step by step, per camera: **Acquire image**, **Detect colors**, **Detect objects**, **Store image**, **Publish to the Data Layer** and the rest. Each camera's normal is

learnt in its first minute. A frame rate below 70 % of normal, or a picture twice as slow, held for ten seconds, is logged as a slowdown that names the step that grew. **Reset max and normal** starts the measurement again.

Accelerators

NVIDIA GPU and **Hailo** say which processor the loaded model runs on, and why. They show the driver version on the device and the versions the snap carries. A model on the CPU when you expected an accelerator is explained here. See System requirements.

Storage

Storage shows what the app has stored on the device: models, saved pictures, the detection history and recordings, with sizes and free disk. Each kind can be deleted. Deleting cannot be undone, and anything in use cannot be deleted.

For support

Download diagnostics (.zip) downloads versions, the GPU and Hailo state, health, timings, the start-up log and the library search paths. It contains no credentials. Send it with every support request; see Support.

Data Layer

A PLC, Node-RED or any ctrlX app uses InSiSo Visual Detector through the ctrlX Data Layer. The app's whole Data Layer is one folder, the same for every cell, whether it has one camera or five and one detector or ten:

```
insiso/visual-detector/
├── ready            BOOL     a trigger will work; if not, message says why
├── message          STRING   "Ready", or what to fix
├── busy             BOOL     a trigger is being processed
├── trigger      (W) STRING   write any value, e.g. your part number
├── picture/                  the picture every block below describes (written last)
│   ├── id                    a counter, one higher per picture evaluated
│   ├── time                  when it was taken, for documentation
│   ├── trigger               what produced it: your value, pulse-<camera>-<n>, or
free-run
│   ├── ok, error             did every camera and detector run
│   └── count                 objects found, all functions and cameras together
├── color/                    Color Detection: every blob, as arrays
├── object/                   Object Detection: every object, as arrays
├── coordinates/              Coordinate Mapping: the same objects in machine mm
├── json             STRING   the whole picture as one text, for Node-RED
├── command      (W) STRING   every web button, by name
└── command-result   STRING   the answer to the last command
```

Is the app on the Data Layer?

Open **Diagnostics**. The first line of chips answers this before anything else: **ctrlX Data Layer**, the licence, each camera and each detector. Its **Trigger** button writes `trigger` exactly as a PLC would. If the button works, the three steps below work from your PLC.

The three steps

Step	Node	What to do
1	`ready`	Wait until it is TRUE. When FALSE, `message` says what to fix, for example *Camera 2 is not connected.*
2	`trigger`	Write any value that changes each time, for example your part counter `1042`. Every connected camera takes one picture and every detector runs once.
3	`picture/trigger`	Wait until it shows `1042`. It is written **last**, so every block already holds this part's answer. Then read what you need.

```
1. wait until  ready           = TRUE
2. write       trigger         := '1042'
3. wait until  picture/trigger = '1042'
   then read   picture/ok, picture/count,
               object/class[0], object/x-px[0], coordinates/x-mm[0], ...
```

Every object is one element of the arrays, and element 0 is the most confident. A colour-sorting cell reads `color/name[0]`. A code-reading cell reads `object/code[0]`. A pick-and-place cell reads `object/x-px`, `y-px` and `orientation-deg`, or `coordinates/x-mm`, `y-mm` and `position-valid` once Coordinate Mapping is set up. A dimensioning cell reads `coordinates/length-mm`, `width-mm` and `height-mm`.

Example, an orange and a red part in view:

```
picture/count     = 2
picture/trigger   = "1042"
color/picture-id  = 4711          (= picture/id)
color/count       = 2
color/name        = ["orange", "red"]
color/x-px        = [412.0, 1180.0]
color/y-px        = [300.0, 310.0]
color/width-px    = [96.0, 102.0]
```

What the answer means

`ok` is not "found something". `ok` says the app did its job: every camera sent a picture and every detector ran. `count` says what was in the picture. An empty belt is `ok = TRUE, count = 0`.

One picture, provably. Values are read one node at a time, so the picture number is on `picture/id`, on every block as `<block>/picture-id`, and on every object as `picture[]`. It is a counter, one higher for every picture the app evaluates. Two values with the same picture number came from the same exposure, whenever they were read. After a restart the counter carries on past its last value, so a number is never given out twice.

Several cameras or detectors. A trigger fires every connected camera at the same moment and runs every detector once, on its own camera's picture. `count` adds up everything found. `camera[]` and `detector[]` say where each element came from. `json` has the same objects in one text:

```
{"trigger": "1042", "ok": true, "count": 1, "objects": [
  {"kind": "object", "detector": "model-1", "camera": 1, "class": "carton",
   "confidence": 0.93, "x_px": 1250.0, "y_px": 980.0, "width_px": 610.0,
   "height_px": 455.0, "orientation_deg": 0.0, "orientation_valid": false,
   "x_mm": 412.5, "y_mm": 120.0, "position_valid": true}]}
```

Orientation. Color Detection measures it from the blob's shape (`orientation-valid` is TRUE when the blob is elongated). A model gives upright boxes, so `orientation-deg` is 0 with `orientation-valid` FALSE unless the model predicts corners.

Which detectors run. Every detector listed under **Is it running?** on Camera Connector. Leave one out by pausing it, or with the command `configure-detector {"id": "model-2", "active": false}`.

Hardware trigger

With the camera in hardware trigger mode, every pulse on its trigger line takes one picture, published exactly as for a software trigger. Since the PLC wrote no value, `picture/trigger` reads `pulse-<camera>-<n>`, *n* being the camera's picture number:

```
1. remember    sLast := picture/trigger
2. pulse       the camera's trigger line
3. wait until  picture/trigger <> sLast
   then read   object/class[0], ...
```

Free run

With free-running cameras, the detectors run all the time and every picture reaches the same nodes, with `picture/trigger = free-run`. They are published when a value changes, at most four times a second. That is for watching, dashboards and simple sorting. A PLC that must read one answer per part uses a trigger.

Commands

Every button in the app's pages is also a command on the `command` node. Write the name, a space and the argument; the answer appears in `command-result`:

```
command := 'set-trigger-mode 1,software'
command := 'camera-status 2'
```

See Data Layer commands for every command.

Structured Text

Function blocks for the three steps (`FB_VisionTrigger`) and for one command and its answer (`FB_VisionCommand`), using the `CXA_Datalayer` library, are available from InSiSo with the PLC acceptance test program. Ask support.

Every node

Generated from the app's node table, version 4.7.21. The app's **Help** page lists those of the installed version. Arrays (`string[]`, `float[]`, `bool[]`) have one element per object.

Node	Type	Access	Content
`ready`	bool	R	TRUE when a trigger will work: licence, cameras and at least one detector are in place. When FALSE, message says why
`message`	string	R	Ready, or what to fix, in plain words
`busy`	bool	R	TRUE while a trigger is being processed

Node	Type	Access	Content
trigger	string	R/W	Write any value, e.g. your part number: every camera takes one picture and every detector runs once. picture/trigger shows the value back, last, when the answer is complete
picture/id	float	R	A counter: the picture every block below describes, one higher for every picture evaluated, whatever camera (with several cameras in one picture, the first camera's). Each block repeats it in its own picture-id. Resumes past the last value after a restart, never repeats. Written last: when it shows a new number, every block already carries that picture
picture/time	string	R	When that picture was taken (UTC ISO 8601), for documentation; several cameras joined with +. Compare picture/id, not this
picture/trigger	string	R	What produced this picture: the value written to trigger, pulse-<camera>-<n> for a pulse on the camera's trigger line, or free-run for a picture the free-running cameras took on their own, or settings-change for a triggered camera's last picture looked at again after its color settings changed. Written last of all: when it equals your value, the answer to your trigger is complete
picture/ok	bool	R	TRUE when every camera delivered a picture and every detector ran. It does not mean something was found; count says that
picture/error	string	R	Why ok is FALSE, in plain words; empty when ok
picture/count	float	R	How many objects every function found in this picture, all cameras together
color/picture-id	float	R	The picture this block describes (equals picture/id once the picture is complete)
color/count	float	R	How many blobs were found: the length of the arrays below
color/name	string[]	R	The color of each blob found, by its name on the Color Detection page; element i of every color/ array is the same blob
color/x-px	float[]	R	Each blob's centre X in the picture, pixels
color/y-px	float[]	R	Each blob's centre Y in the picture, pixels
color/width-px	float[]	R	Each blob's width in the picture, pixels
color/height-px	float[]	R	Each blob's height in the picture, pixels
color/orientation-deg	float[]	R	Each blob's orientation from its shape, degrees; meaningful only where orientation-valid is TRUE
color/orientation-valid	bool[]	R	TRUE where the blob's shape gives an orientation (elongated), FALSE for a round or square blob
color/confidence	float[]	R	How well each blob fits its color range, 0..1
color/camera	float[]	R	The camera that saw each blob
color/picture	float[]	R	

Node	Type	Access	Content
			The picture counter each blob was found in; differs from picture-id only when several cameras contribute to one picture
object/picture-id	float	R	The picture this block describes (equals picture/id once the picture is complete)
object/count	float	R	How many objects the models found: the length of the arrays below
object/class	string[]	R	The class of each object the models found; element i of every object/ array is the same object
object/x-px	float[]	R	Each object's centre X in the picture, pixels
object/y-px	float[]	R	Each object's centre Y in the picture, pixels
object/width-px	float[]	R	Each object's box width in the picture, pixels
object/height-px	float[]	R	Each object's box height in the picture, pixels
object/orientation-deg	float[]	R	Each object's orientation, degrees, when the model provides one; 0 with orientation-valid FALSE for a model that gives upright boxes
object/orientation-valid	bool[]	R	TRUE where the model measured the object's orientation
object/confidence	float[]	R	Each object's confidence, 0..1
object/camera	float[]	R	The camera that saw each object
object/detector	string[]	R	The detector that found each object, e.g. model-1
object/picture	float[]	R	The picture counter each object was found in; differs from picture-id only when several cameras contribute to one picture
object/code	string[]	R	The Data Matrix code read inside each object; empty where none was read
coordinates/picture-id	float	R	The picture this block describes (equals picture/id once the picture is complete)
coordinates/count	float	R	How many objects are listed: the length of the arrays below, the same objects as object/ in the same order
coordinates/class	string[]	R	The class of each object, as object/class; element i is object i
coordinates/x-mm	float[]	R	Each object's machine X (Coordinate Mapping)
coordinates/y-mm	float[]	R	Each object's machine Y (Coordinate Mapping)
coordinates/position-valid	bool[]	R	TRUE where x-mm and y-mm are measured; FALSE when that camera is not calibrated (Coordinate Mapping page)

Node	Type	Access	Content
`coordinates/length-mm`	float[]	R	Each object's length (longer side); 0 where not measured
`coordinates/width-mm`	float[]	R	Each object's width (shorter side); 0 where not measured
`coordinates/height-mm`	float[]	R	Each object's height; 0 where not measured
`coordinates/camera`	float[]	R	The camera that saw each object
`coordinates/picture`	float[]	R	The picture counter each object was found in
`json`	string	R	Everything this picture holds, every object from every camera and function, as one JSON text (for Node-RED or a PLC that parses JSON)
`command`	string	R/W	Write a command by name, then a space and its argument, e.g. "set-trigger-mode 1,software". Every button in the web pages has one; the Help page lists them. The outcome is in command-result
`command-result`	string	R	JSON outcome of the last command. Starts `{"ok": ..., "command": ..., "argument": ...}` so a PLC sees both within 255 characters

Data Layer commands

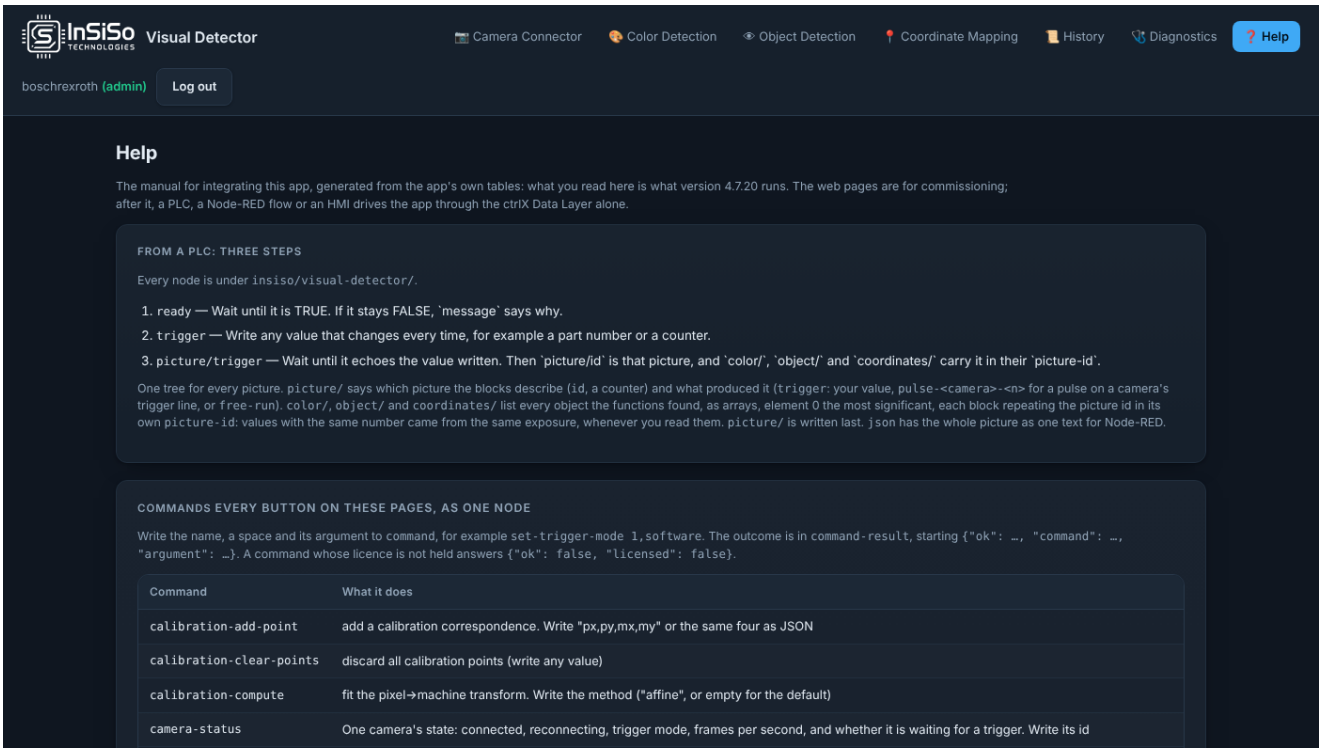
Every button in the app's pages can also be pressed by a PLC, Node-RED or another ctrlX app, through one Data Layer node:

```
insiso/visual-detector/command         write: the name, a space, the argument
insiso/visual-detector/command-result  read:  the answer, as JSON
```

For example `set-trigger-mode 1,software`, or `configure-detector {"id": "model-1", "active": false}`. The argument is the same JSON or comma-separated text the button sends. Leaving a field out gives the button's default.

`command-result` starts `{"ok": ..., "command": ..., "argument": ..., "at": ...}`, so a PLC's 255-character string functions see it. With `"ok": false`, `"error"` says why in plain words. A command whose function is not licensed also says `"licensed": false`. The answer belongs to your write once it names your command and its `at` differs from before you wrote. Commands run one at a time, in the order written.

Writing `list-commands` returns every name. The tables below are the commands of version 4.7.21, taken from the app's own command table. The app's **Help** page always lists those of the installed version, including the parcel cell commands of the Parcel Tracking licence, which this manual does not describe.



Cameras

Command	What it does	Licence
`connect-camera`	Connect a camera. Write the source — a USB index ("0"), "genicam:<n>", or an rtsp:// / http:// URL — or JSON `{"id": 2, "source": "0"}` to put it in a particular slot. Without an id it takes the next free one	Camera Connector

Command	What it does	Licence
`disconnect-camera`	Disconnect a camera. Write its id ("2") or JSON `{"id": 2}`; write any non-numeric value for the selected camera	Camera Connector
`scan-cameras`	Scan for cameras (write any value); command-result lists what was found	Camera Connector
`trigger-camera`	Take one picture now on one camera and run its detectors. Write a camera id ("2"); the answer is in command-result. To get picture/ and the blocks, write trigger	Camera Connector
`trigger-all-cameras`	One picture on every connected camera at the same moment, every detector once. command-result lists each camera's frame number and the spread between their exposures (skew_ms). The `trigger` node does the same and also publishes the picture	Camera Connector
`set-trigger-mode`	Set when a camera produces pictures. Write JSON `{"id": 2, "mode": "software"}` or "2,software". Modes: free-run (every cycle), software (one per trigger), hardware (the camera's own trigger line; GenICam cameras only)	Camera Connector
`save-picture`	Save a picture now. Write a camera id ("2") or any other value for the selected camera; the file lands in the device's capture buffer	Camera Connector
`saved-pictures-config`	Configure saved pictures. Write JSON with any of on_detection, classes, mode (raw|annotated|both), max_captures, max_megabytes, min_interval_seconds, jpeg_quality, folder ("" for the device's storage, or an absolute path on a mounted drive under /media, /mnt or /run/media), every_picture (true saves every picture, at most one every min_interval_seconds, and only to a folder on a mounted drive)	Camera Connector
`delete-saved-pictures`	Delete saved pictures. Write a camera id ("2") or any other value for all of them	Camera Connector
`camera-status`	One camera's state: connected, reconnecting, trigger mode, frames per second, and whether it is waiting for a trigger. Write its id	Camera Connector

Detectors

Command	What it does	Licence
`create-detector`	Create a detector instance. Write `{"kind": "model", "camera": 1}` or "model,1"	Runtime
`delete-detector`	Delete a detector instance. Write its id	Runtime
`configure-detector`	Change a detector. JSON with its id and any of model, camera, active, confidence, read-codes, code-depth, e.g. `{"id": "model-1", "active": false}`	Runtime
`pause-detector`	Pause one detector, as unticking Detect objects on Object Detection does. Write its id, e.g. model-1. Not remembered: resume-detector or the next restart resumes it	Runtime
	Resume a paused detector. Write its id, e.g. model-1	Runtime

Command	What it does	Licence
`resume-detector`		
`detector-status`	One detector's state: camera, model, active, paused, and what it runs on (cpu, cuda, tensorrt or hailo). Write its id, e.g. model-1	Runtime
`select-model`	Load a model by file name on the Object Detection page	Object Detection Model
`detect-once`	Run one detection now. Write "color" or "model" to choose the detector, or any other value to use the loaded model if there is one and color detection otherwise	Object Detection Model
`start-stream`	Start continuous model inference. Write the source — a USB index ("0") or an rtsp:// / http:// URL	Object Detection Model
`stop-stream`	Stop continuous model inference (write any value)	Object Detection Model
`set-confidence`	Set the detection confidence threshold. Write a number between 0 and 1	Object Detection Model
`set-classes`	Rename the model's output classes. Write a comma-separated list in class-index order	Object Detection Model
`set-class-overlap`	One object, one box. Write "on" to drop a box that overlaps a more confident box of a different class, "off" to report both, or the overlap threshold (0.05-0.95, default 0.6) to switch it on at that value. Applies to every model on the device and is remembered across restarts	Object Detection Model
`reset-timings`	Zero the performance statistics including min/max (write any value)	Object Detection Model

Color

Command	What it does	Licence
`create-color`	Add a color to a color detector. Write JSON `{"id": "color-1", "label": "carton", "h": [10, 30], "s": [80, 255], "v": [70, 255]}` or "color-1,carton". It is appended, so existing colors keep their addresses	Color Detection
`delete-color`	Remove a color. Write JSON `{"id": "color-1", "color": 2}` or "color-1,2". The colors after it move down one address	Color Detection
`color-config`	Apply color settings. Write JSON with any of ranges, min_area, max_detections, blur	Color Detection

Calibration

Command	What it does	Licence
`calibration-add-point`	Add a calibration correspondence. Write "px,py,mx,my" or the same four as JSON	Position / Coordinate Mapping
`calibration-clear-points`	Discard all calibration points (write any value)	Position / Coordinate Mapping
`calibration-compute`	Fit the pixel->machine transform. Write the method ("affine", or empty for the default)	Position / Coordinate Mapping
`convert-to-mm`	Convert a pixel coordinate to machine mm. Write "px,py" or JSON `{px, py}`	Position / Coordinate Mapping
`convert-last-detection`	Convert the most confident recent detection's centre to machine mm (write any value)	Position / Coordinate Mapping

Detection history

Command	What it does	Licence
`history-mark`	Write the current detections to the detection history now. Write any value, or a short note to store with the row	Detection History
`history-config`	Configure the detection history. Write JSON with any of enabled, max_rows, max_megabytes, move_px, heartbeat_seconds, storage_path. storage_path is where the database is kept: empty for the device's own storage, or a directory under /media, /mnt or /run/media for a mounted drive. Changing it moves the existing history with it	Detection History
`history-clear`	Delete every recorded detection (write any value). Cannot be undone — export the CSV first	Detection History

Live previews and the app

Command	What it does	Licence
`start-detection`	Start the headless detect-and-publish loop (write any value)	InSiSo Visual Detector (base runtime)
`stop-detection`	Stop the headless detect-and-publish loop (write any value)	InSiSo Visual Detector (base runtime)
`datalayer-verify`	Read every node back from the device and report what it holds (write any value)	InSiSo Visual Detector (base runtime)
`list-commands`	Every command name	Runtime
`performance-status`	Each camera's frame rate and the loop's cycle time per stage, against the normal learnt in its first minute, with the last slowdown and what	Runtime

Command	What it does	Licence
	grew (write any value). The watchdog logs a slowdown held for ten seconds	
`reset-performance`	Clear every camera's maximum processing times and its normal, which is then learnt again over the next minute (write any value). Answers like performance-status	Runtime

Diagnostics (free)

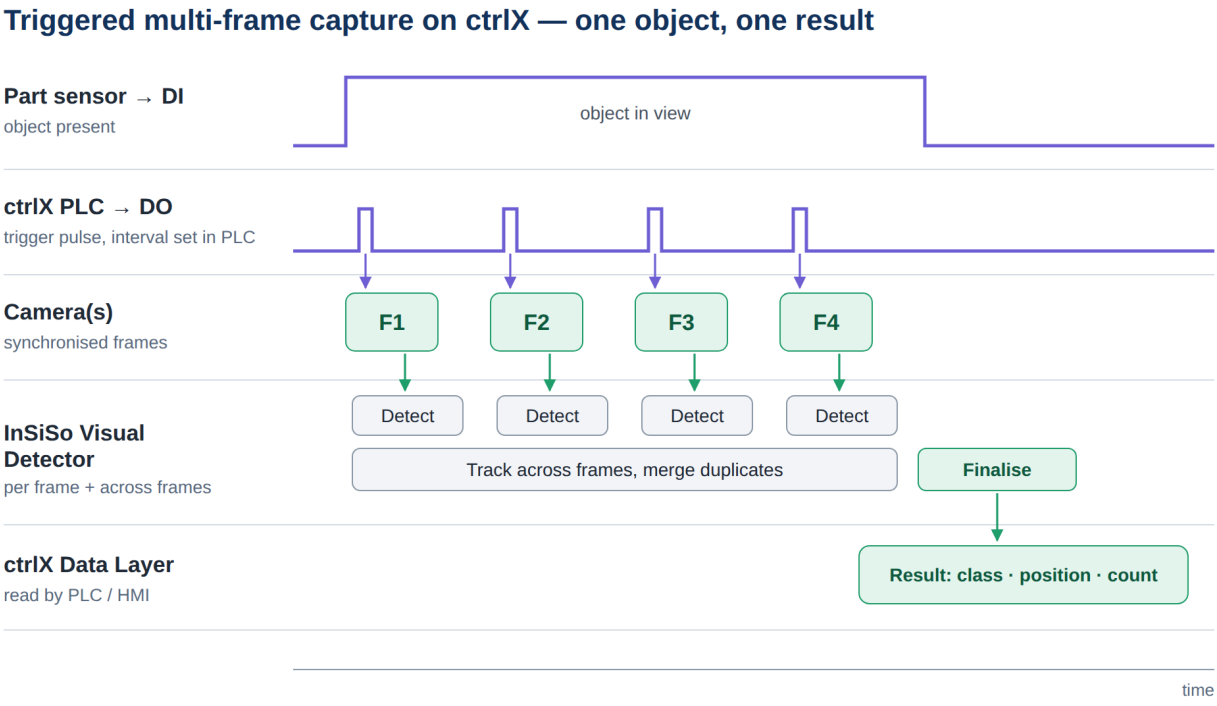
Command	What it does	Licence
`recorder-start`	Start recording. JSON as the page sends it: `{"tof_camera", "label_camera", "root", "tag", "window_ms", "every_s", "max_gb", "make_3d"}`	Diagnostics
`recorder-stop`	Stop recording (write any value)	Diagnostics
`recorder-tag`	Tag what is being recorded. Write the tag	Diagnostics
`recorder-sessions`	List recorded sessions. Write any value, or JSON `{"root": "<USB drive>"}`	Diagnostics
`recorder-delete`	Delete a recorded session. Write its name, or JSON `{"name", "root"}`	Diagnostics
`storage-list`	What the app has stored, by kind, with sizes and free disk (write any value)	Diagnostics
`storage-delete`	Delete one stored item. Write "kind,name" or JSON `{"kind", "name"}`. Kinds: models, pictures, recordings, label-bench, uploads, videos, parcel-cell, history ("history,detection-history.db" clears the Detection History). Anything in use is refused, with why	Diagnostics
`storage-delete-all`	Delete every item of one kind that is not in use. Write the kind	Diagnostics

Captures for retraining

Pictures saved on the device go back to InSiSo Model Bench as training data, so a model can be corrected with frames from the real line.

1. On **Camera Connector**, under **Saved pictures**, set **Save** to **Raw camera frame (for re-training)** or **Both**, and switch on **Save automatically when something is detected**, or press **Take a picture now**. See Saved pictures.
2. Press **Download all (.zip)**. You get `captures.zip`: every saved picture with a JSON file of what was detected in it.
3. In Model Bench, choose **Import ctrlX Captures**. The frames arrive as unreviewed annotations under `ctrlx/<device>/cam<N>`.
4. Correct, mark as reviewed, retrain and export the new model. Load it on the device as in Object Detection.

Importing the same zip twice adds nothing.



Test setup

How to set up InSiSo Visual Detector on a ctrlX CORE for a typical usage and test scenario. It is written for Bosch Rexroth's app validation and works for any commissioning check. The two Postman collections it refers to, `standard-scenario1.json` and `standard-scenario2.json`, come with the app's validation package.

What you need

Item	Needed for
ctrlX CORE X3 (arm64 snap), or ctrlX CORE X5, X7 or ctrlX CORE virtual (amd64 snap), on ctrlX OS 4.x	Everything
The **Runtime** licence, installed or as a trial	Everything
A PC with a browser and Postman	Running the scenarios
A GenICam / GigE Vision or USB / UVC camera, the **Camera Connector** and **Color Detection** licences	Scenario 2 only

Scenario 1 needs no camera and no model.

Set up the device

1. Install the snap for the device's architecture: see Installation.
2. Install the licences, or start their trials, in **Settings ▸ Apps & Licenses**.
3. Open **InSiSo Visual Detector ▸ Diagnostics**. The **Licence** card lists each function as **unlocked** or names its licence. The first chip, **ctrlX Data Layer**, shows ✔.
4. In Postman, import both collections. Set the collection variables `baseURL` (for example `https://192.168.1.1`), `username` and `password` to the device's address and a ctrlX OS user.

Scenario 1: no peripherals

`standard-scenario1.json`. It changes no setting and can be run any number of times.

1. Signs in with the ctrlX OS user (identity manager) and keeps the token.
2. Reads `ready`, `message` and `busy` under `insiso/visual-detector/`.
3. Runs the status commands through `command` and reads each answer in `command-result`: `list-commands`, `scan-cameras`, `performance-status`, `storage-list`.
4. Writes `postman-scenario-1` to `trigger` and reads what it published: `picture/id`, `picture/trigger`, `picture/ok`, `picture/error`, `color/count`, `object/count`, `coordinates/count` and `json`.
5. Calls the REST API the pages use: licences, health, performance, cameras, models and the detection history.

Expected: every call answers HTTP 200. Each command's answer names its command. With no camera, `ready` is FALSE and `message` says why; the trigger publishes `picture/ok` FALSE with the reason in `picture/error`.

Scenario 2: one camera

`standard-scenario2.json`. Connect one GigE Vision or USB camera to the device first.

1. Signs in.
2. `scan-cameras` finds the camera and keeps its address in the variable `cameraSource`. To pick another camera, set `cameraSource` by hand.
3. `connect-camera` connects it as camera 1. `set-trigger-mode 1,software` makes it take a picture only when triggered. `camera-status 1` reads its state.
4. `create-color` adds red to the camera's color detector, `color-1`.
5. Writes `trigger` twice and reads the results of each picture: `picture/id`, `picture/trigger`, `picture/ok`, `color/count`, `color/name`, `color/x-px`, `color/y-px`, `object/count` and `json`.
6. `detector-status color-1` reads the detector's state.
7. `set-trigger-mode 1,free-run` leaves the camera running and the detector evaluating every picture: the state a commissioned cell runs in.

Expected: `ready` is TRUE from step 4 on. Each trigger raises `picture/id` by one and echoes its value in `picture/trigger`. With a red part in view, `color/name` lists `red` and `color/x-px` and `color/y-px` give its centre.

Power cycle

Power-cycle the device without signing in. Within three minutes the app runs, the camera is reconnected with its trigger mode, the detectors run, and `ready` is TRUE. `picture/id` carries on past its last value. Nobody needs to press anything.

Clean up

Write `disconnect-camera 1` to `command`, or press **Disconnect** on Camera Connector. **Reset colors** on Color Detection removes the red added by scenario 2.

FAQ

Does Visual Detector need a cloud connection?

No. Detection runs on the ctrlX device. Pictures, models and results stay in your network.

Which cameras work?

GenICam / GigE Vision industrial cameras, USB / UVC cameras, and RTSP or HTTP streams. Each connected camera needs one Camera Connector licence. See Camera Connector.

How does the PLC read results?

From the ctrlX Data Layer, in three steps: wait for `ready`, write `trigger`, read when `picture/trigger` shows your value. See Data Layer.

Do I need a model?

No. Color Detection finds parts by colour with no model. A model from InSiSo Model Bench is needed for Object Detection.

Can it give pick positions to a robot?

Yes, with the Machine Coordinate Transformation licence. See Coordinate Mapping.

What happens after a power cut?

The app starts by itself, reconnects its cameras, loads its model and detects, with no button pressed. The picture counter carries on where it stopped.

Why does the model run on the CPU when the device has a GPU or a Hailo module?

Diagnostics ▸ Accelerators says why, usually a driver version the snap does not carry. See Accelerators.

Why does the app show a login page?

ctrlX OS did not pass your sign-in to the app; the box above the login says why. Open the page from the ctrlX OS sidebar, not from a bookmark or a typed address. If the login still appears, log out of ctrlX OS, log in again and open the page from the sidebar. See Sign in.

> **Why does `ready` stay FALSE?**
>
> `message` says what to fix, for example a camera that is not connected or a licence that is not held. **Diagnostics** shows the same in its chips.

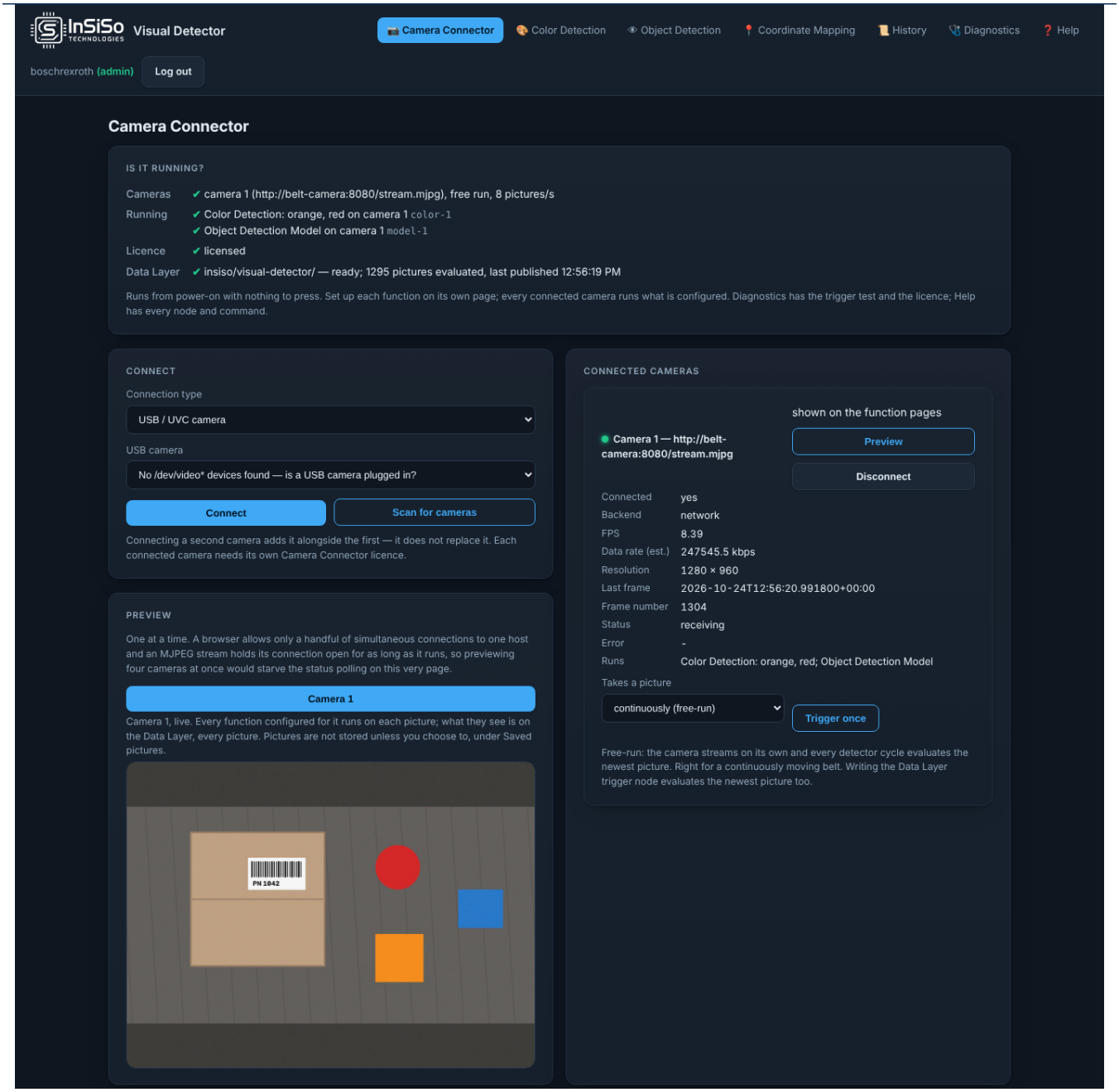
> **What should I send to support?**
>
> The file from **Diagnostics ▸ Download diagnostics (.zip)**, the ctrlX device and ctrlX OS version, and what you did. See Support.

Release notes

The current release is **4.7.21** (October 2026), the first release of InSiSo Visual Detector for the ctrlX Store. It replaces 4.7.20, which went to Bosch Rexroth's app validation and was not released.

What’s in 4.7.21

- **Version numbers follow ctrlX OS.** The first number is the ctrlX OS major version: 4.x.x runs on ctrlX OS 4.x. Test builds before 4.7.18 had higher numbers, up to 7.2.17; they are all older.
- **Two snaps.** arm64 for the ctrlX CORE X3; amd64 for the ctrlX CORE X5 and X7, the ctrlX IPC and ctrlX CORE virtual. Both carry Hailo-8 support; the amd64 snap also carries the NVIDIA GPU runtime.
- **Camera Connector.** GenICam / GigE Vision, USB / UVC, RTSP and HTTP cameras, several at once. Free run, software trigger or the camera's hardware trigger line. **Is it running?** answers on one card. Saved pictures, to the device or a USB drive.
- **Color Detection.** Seven preset colours, all off until ticked, and colours sampled from the picture. One minimum blob area for every colour. The page shows exactly what the Data Layer got.
- **Object Detection.** Models from InSiSo Model Bench: `.onnx` on the CPU or an NVIDIA GPU, `.hef` on the Hailo-8 module. Class names and class colours come with the model. Pause per camera.
- **Code Reading, 3D Dimensioning, Coordinate Mapping.** The Data Matrix in each detected label; parcel length and width from a corner-trained model; positions in machine millimetres.
- **One Data Layer folder.** `insiso/visual-detector/`: `ready`, `trigger`, `picture/`, `color/`, `object/`, `coordinates/` and `json`, every object as arrays, every value with the picture number. Every button is a Data Layer command.
- **Nobody presses anything after commissioning.** After a reboot the app reconnects its cameras, loads its model, takes its licence seats and detects.
- **Diagnostics.** A trigger test, the licence, performance per step with a frame-rate watchdog, accelerators, storage, and one zip for support.
- **Detection History**, with CSV export and size limits, on the device or a USB drive.
- **Restart delay.** If the app stops unexpectedly, the device restarts it after 10 s and keeps doing so, so it comes back by itself.
- **Signed in from the sidebar.** The sidebar links carry the ctrlX OS sign-in to the app, as the ctrlX AUTOMATION SDK documents. 4.7.20 relied on a ctrlX OS browser cookie that not every device sends, and showed its own login there.



All releases

Version	Date	Highlights
4.7.21	October 2026	First release. Camera Connector, Color Detection, Object Detection, Code Reading, 3D Dimensioning, Coordinate Mapping, Detection History and Diagnostics, on one Data Layer folder.

Known limitations in 4.7.x

- **TensorRT** is not included. **NVIDIA GPU (TensorRT)** runs the model on CUDA.

- **NVIDIA GPU** works only with the NVIDIA driver versions the amd64 snap carries: those Ubuntu 24.04 has shipped from 535 to 580. On another version the model runs on the CPU, and Diagnostics shows the version.
- **Hailo-8** works only with Hailo driver 4.23.0, and 4.24.0 on the arm64 snap. On another version `.hef` models do not load, and Diagnostics shows the version.
- **Code Reading** reads Data Matrix codes. It is switched on per detector with the Data Layer command `configure-detector`; there is no switch on the page yet.
- **3D Dimensioning** uses a fixed assumed parcel height unless a measured height is supplied. No accuracy figure is given.
- **Orientation** from a model is 0 (`orientation-valid` FALSE) unless the model predicts corners.
- **Free run** publishes at most four pictures a second. Use a trigger for one answer per part.

Workarounds

- **After an update, reboot the device once.** This frees licence seats the previous version still held.
- **An interface not connected** on an app installed from file: Diagnostics names it. Connect it as in Installation, then restart the app.

Known defects

No defects are open against 4.7.21 at the time of release. Report one to support with the diagnostics zip.