Visual Registration
Wim Luyendijk
445662

Colophon

Date	April 9, 2023
Version	1.0
Department	Research and Development
Author	Wim Luyendijk

Table of Contents

List of Figures

Summary
Movella, a leading innovator in digitising movement, is a key provider of Micro -Electro - Mechanical -Systems (MEMS) sensors. Currently, the sensors are registered in a data - base using handheld scanning devices in a two-step process. First, during testing and calibration of the sensors and second, prior to shipment and distribution of the product. Movella is interested in developing and testing a computer vision system to automatically register the product. The company's ultimate goal is to register each sensor prior to testing and calibration. This project describes the design, development, demonstration, testing and evaluation of a conceptual computer vision system that Movella can implement for product registration prior to shipment and distribution "as is" or can adapt to automatically register the product prior to testing and calibration.

1 Introduction

Movella in Enschede calibrates and packages industrial motion trackers (MTi) which uses small MEMS (Micro-Electro-Mechanical-Systems) inertial sensors to measure changes in motion. The smallest motion tracker, the MTi-1 series is sold in large quantities and is sourced from different companies and locations. The devices are shipped on trays and each device has it's own barcode label known as a Matrix Code which contains its serial number and product information. Currently all individual devices are scanned manually which is a labour-intensive process.

Movella has created an assignment for this internship to create an automated scanning solution which recognizes and reads the Data Matrix (DM) codes of the devices using Computer Vision (CV). Besides the development and testing of the algorithm, the internship also includes the design of a robust and easy-to-operate hardware setup. Movella has given the student (Wim Luyendijk) the freedom to investigate different approaches using any technology available on the market.

Like Movella, many manufacturing companies are constantly looking for tools to increase operational efficiency. One tool which consistently delivers increased efficiency and productivity across a broad range of industries is the barcode label (Stazzone, 2023). While there are many types of barcodes, these labels basically fall into two categories: linear codes (Figure 1), which tie the code to product information in an external database, and matrix codes, similar to Quick Response Codes, most commonly known as QR codes (Figure 1). These codes can be used for a wide range of applications from referencing a store price to linking to a company's website. In Movella's case, they are using the code to store ID & production information to facilitate backtracking to find sensor information if it is needed in the future.

9 771234 567003

Figure 1 Barcode from WorldBarcodes.com

Figure 2 QR code from W3Programmings.com

To facilitate tracking and processing, barcodes are encoded with a products information in alphanumeric characters. The codes are then read by either a handheld device or one built into a checkout station. When paired with the right barcode scanning system, barcode labels lend themselves to improve data collection processes by maintaining consistent, efficient workflows that positively impact the organisation and the supply chain (Stazzone, 2023). The main application of barcodes in organisations are to track goods throughout their life cycle, from manufacture to distribution to purchase to service and repair (McCue, 2022). One company that uses barcodes to keep track of their product during its calibration and packaging process is Movella.

1.1 Movella

With offices in North America, Europe and Asia, Movella is a leading innovator in digitising movement and a key provider of MEMS sensors (Movella About Page, 2023). For this, the company designs and develops a series of advanced motion sensors that over time have found their way to a variety of industrial, sports, healthcare, and mobility markets, including the entertainment industry. In the latter multiple Movella motion sensors are placed on the human body to capture human movement in order to digitize movement for games, such as the popular soccer simulation game FIFA, and movie animations like Avatar, Mandalorian, The Matrix, Disney’s The King’s Man and others.

Located in Kennispark, Enschede, Movella’s branch office in The Netherlands targets the development of 3D motion technology, ranging from full-body 3D kinematic solutions to 3D motion trackers to capture the movement of industrial systems and people. The sensors used to estimate the motion of interest use the high-rate measurements of IMUs combined with information from other sensors such as magnetometer, GNSS, barometer in so-called sensor fusion algorithms (Sensor Modules, 2023).

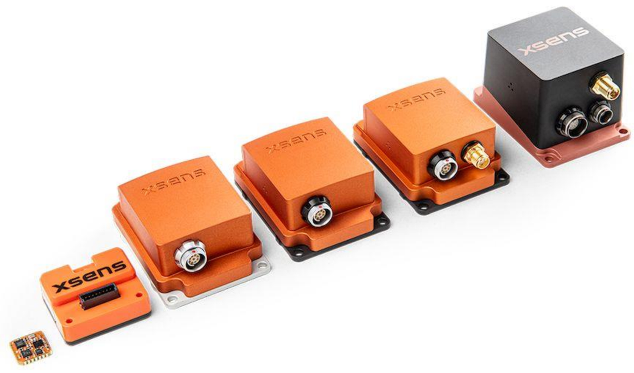


Figure 3 MTi product line

Figure 3 above, depicts the portfolio of industrial motion trackers. Depending on the sensors and algorithm running on each product platform the output of the sensor ranges from orientation to a full dynamic state estimate which includes position and velocity estimates. The sensors shown are the MTi 1 surface-mount device module series (far left), the MTi 600 easy integration series (2nd from left), the MTi 10 series entry level model with robust accuracy (3 middle sensors), and the MTi 100 high-end class of MEMS IMUs, orientation and position sensor modules (far right). All products contain a 3D IMU composed of a gyroscope and an accelerometer plus a 3D magnetometer, with optionally a barometer and GNSS receiver (Mti Product Selector, 2023). From Movella's product line the focus of the project is the MTi-1 sensor due to it's high demand.

1.1.1 MTi-1 Sensor

The MTi-1 series sensors (Figure 4). With a footprint of just 12.10 x 12.10 x 2.55 mm, and weighing < 1 gram, (MTi-1 IMU, 2023) the sensors fit a wide range of (embedded) applications, including unmanned aircraft and drones, surface and underwater robotics, autonomous vehicles, and indoor mobile robots.

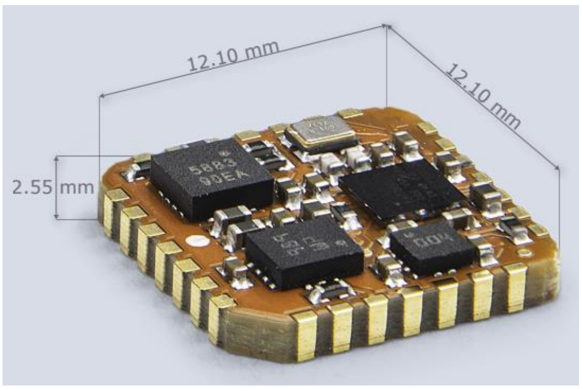


Figure 4 Close up image of the MTi-1 Sensor

Movella receives these sensors pre-assembled with the bar code already set on it, so they can begin the calibration process without delay. During the calibration process, the sensors are registered to Movella's internal database using the Barcodes on each of the sensors. Where the sensors can be followed on their progress through calibration and be counted towards the inventory of the sensors.

1.1.2 MTi-1 Barcodes

The barcode labels chosen by Movella for the MTi-1 modules are Data Matrix codes. The reason DM codes, instead of one-dimensional bar codes are chosen is related to both the footprint and the amount of information that can be stored on the label. One Dimensional labels have only vertical or horizontal lines and typically store only 9-13 characters, while 2D barcodes can have both creating a matrix of black and white cells, allowing them to hold as many as 2,335 characters (Data Matrix, 2023).

The high density of characters further reduces the footprint of the code thereby making them ideal for small items with minimal space for barcodes, like MTi-1 sensors. DM codes have the added benefit of robust error-checking capabilities. Allowing scanners to still read the codes even when they have significant damage (Data Matrix, 2023).

Movella uses the DM codes to store the MTi-1 sensor Manufacturer ID and traceability information. Figure 5 and 6 below show an example of a DM barcode. The DM code has an "L" shape on the bottom left corner of the code, this is called the "Finder Pattern". On the top right there is another "L" shape of alternating squares, this is the "Timing Pattern". The main square is where the data of the Matrix Code is stored in bytes. Where the black cells represent a 1 and the white cells represent a 0 in binary. (Karrach, 2021).

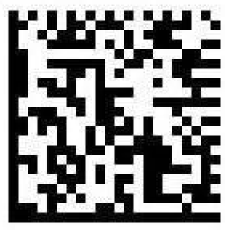


Figure 5 Example Data Matrix

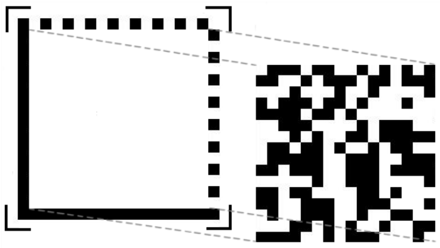


Figure 6 Example of Data Matrix Sections

Using a specific 2D scanning device (Appendix A.1 Two Dimensional Barcode Reader), the code can be decoded and read to register the product. However, reading the data from the data section can also be done by hand or decoded by a program. For more information on how DM codes can be read, see Appendix A.2 Reading Data Matrix Codes.

1.2 The problem

Movella currently uses a handheld laser gun to read the Matrix Code on each of the MTi-1 sensors to register the product. While the operators are fast at scanning, registering the sensors takes on average about 30 seconds for a tray of 20 sensors. After interviewing production operators an import problem was brought to light.

During the scanning process, a matrix code may be missed and not registered to the system. When this occurs the only feedback, the operators receive from the current system is that they do not have the expected number of sensors. This is an issue because the operators do not know which sensor was unable to be registered. The current solution is to rescan the entire tray of individual sensors until the expected number of registered sensors is met. Should a DM code have any issues that cause it to be unreadable, the problem will then cause significant delays. Causing an already labour-intensive and repetitive task to become more time consuming.

To solve the issues of not knowing which sensor is unregistered, the solution shows the operator where the sensor(s) with problems are so it can be addressed individually, which will save the operators time and reduce the amount of repetitive tasks.

1.3 Company Goals

Movella is interested in addressing the DM scanning code problems identified and would like to see a system in place that can automatically scan and register a full tray of sensors and identify problematic sensors and pinpoint faulty or non-detectable codes in real time. It is expected that the designed and developed system can register multiple sensors at one time in several processes, meaning registration of 20 sensors per tray and then 100 sensors per tray. The ultimate goal is to apply the system to all incoming pre-assembled MTi-1 modules and get rid of manual and individual scanning processes. For this the system will need to be highly reliable.

1.4 Stakeholders

Project stakeholders include Movella, Saxion University of Applied Sciences, and the author of this document. Movella, in particular the Production Department and the Research and Development (R&D) Division, is the primary stakeholder. The reason is that both directly benefit from the project's outcome and results. Saxion University of Applied Sciences (SUAS) is the secondary stakeholder as the institution maintains a close and professional relationship with Movella. Finally, the author of this document (Wim Luyendijk) is the tertiary stakeholder as he is a student at SUAS and performed the work in the form of a graduation internship to fulfil part of the requirements to obtain a degree in Software Engineering.

1.5 Project Goals & Requirements

Given the problem outlined in section 1.2, Movella is looking for a solution that will increase the efficiency of registering sensors. This can be done by reducing the amount of time it takes to scan a tray of sensors and remove the obscurity of which sensor is unable to be registered. The solution will also need to maintain the same level of correctness as the current solution when registering the sensors. Meaning for every sensor read, the ID entered in the system must be correct.

During a meeting with Movella's stakeholders, we identified together what the requirements of the project will be to reach a minimum viable product.

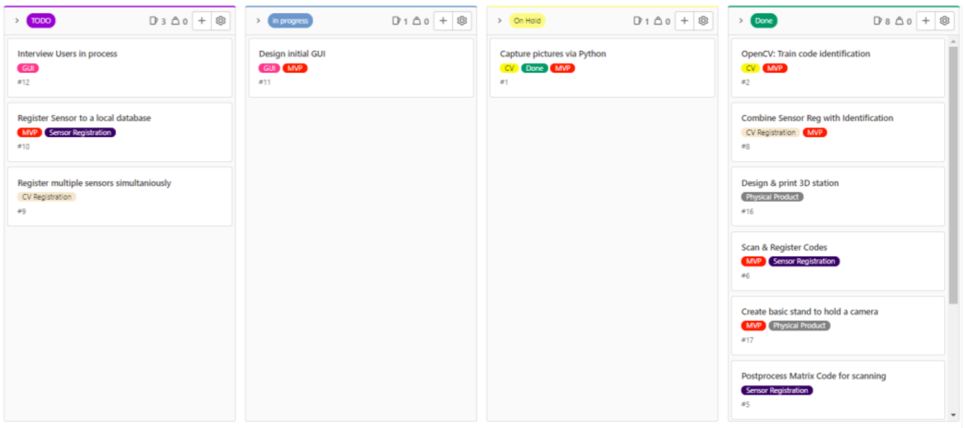
1.5.1 Minimum Viable Product*:*

- The program can detect the positions of multiple Matrix Codes within an image.
- The program can read the data of a Matrix Code with 100% correctness.
- The system can process a tray of sensors without registering the same code twice.
- There is a GUI an operator can use for the process, to see if action is required.
- The system can reliably register the sensors and is robust enough to handle missing Matrix Codes, crooked sensors (sensors not properly set on trays), and other outlying situations.
- The system takes the same amount of time or less than a trained operator scanning a tray (30 seconds for a tray of 20).

2 Project Methodology

The project uses four different tools to ensure completion of all tasks and delivery of the agreed upon product. Tool 1 is a project management methodology (Kanban) that was adopted to facilitate the tracking and monitoring of all project tasks. Tool 2 provides access to Movella's online Confluence system which ensures project documentation can be uploaded and company employees can see informed about project progress and final product delivery. With Tool 3, weekly meetings were organised between all stakeholders (TEAMS was used as an additional tool for unplanned and quick communication) to discuss project progress and challenges. Finally, towards the end of the internship, a change in methodology was made to adapt a SCRUM approach to ensure project tasks stayed in focus and delivery of the product stayed on track. Following is a detailed description of these four project tools and their accomplishments.

<u>Tool 1:</u> Kanban approach.
Kanban is an Agile management approach to visualise workloads and workflow so one can stay up to date on who is doing what work and what stage the work is in (Kanban Methodology, 2023). To achieve this, a so-called Kanban board is generated which displays all the work that needs to be done in a project board organised by columns with each column representing a stage of work. A task then moves from right to left through the columns until it is completed.



TODO 3 0
Interview Users in process
GUI
#12
Register Sensor to a local database
MVP Sensor Registration
#10
Register multiple sensors simultaniously
CV Registration
#9

in progress 1 0
Design initial GUI
GUI MVP
#11

On Hold 1 0
Capture pictures via Python
CV Done MVP
#1

Done 8 0
OpenCV: Train code identification
CV MVP
#2
Combine Sensor Reg with Identification
CV Registration MVP
#8
Design & print 3D station
Physical Product
#16
Scan & Register Codes
MVP Sensor Registration
#6
Create basic stand to hold a camera
MVP Physical Product
#17
Postprocess Matrix Code for scanning
Sensor Registration
#5

Figure 7 Screenshot of kanban board in GitLab

For this project, a four column Kanban board was generated to track and monitor tasks from the initial idea-phase into the steady flow of work phase. The columns were labelled “to do”, “in progress”, “on hold”, and “done” (Fig. xx). By placing cards, representing the workflow phase of each task in the columns, the board proved useful in visualising where each task was in real time. This helped in balancing the workload and in identifying workflow challenges.

It also helped in reprioritizing tasks that were “stuck” to make sure each task was completed on time. To initialise work, each task was prioritised, and placed on the Kanban board to track and monitor the tasks through the stages. Each task then moved through the columns until the task was completed. To use the Kanban approach, Gitlab (an online DevOps tool), was used to order, create, and move the various tasks through the Kanban pipeline. The project’s code was stored in Movella’s Gitlab’s online repository, where it is updated and uploaded between tasks. In the case of unexplainable errors, the code can be reverted back to the previous stable version stored in the repository.

Tool 2: Confluence
Confluence is a collaboration wiki tool used to help teams share information and knowledge. For this project, the software was used to upload project documentation and to share project information like daily work logs (B.1 Work Log), weekly meeting notes (B.2 Weekly Meeting Notes), and document updates among company employees. In so doing, Movella guarantees that those interested in the project not only are kept in the loop of what progress their co-workers are making but are also given an opportunity to actively participate on tasks and thus on the development of the product.

Tool 3: Teams
Microsoft (MS) Teams was used as a communication tool to reserve meeting rooms, hold virtual meetings as needed, and to ask questions which did not need the input of all stakeholders. Teams thus facilitated quick communication between individual stakeholders. To facilitate communication and/or discussion of project related items or events, weekly stakeholder meetings were scheduled. These meetings typically were between the intern and Movella employees or (biweekly) between the intern and the Saxion University representative. During these (bi)weekly meetings, work progress using the Kanban board was discussed along with challenges or problems encountered in executing the tasks. Other points of discussion were the reprioritization of tasks, the review of (updated) documentation, and comments received from Confluence user project participants.

Tool 4: SCRUM
While the Kanban approach proved useful in managing the movement of individual tasks throughout the project when the time tasks took to do were unknown. As tasks became more specific, and the timely completion of tasks became more important, the decision

was made to replace the Kanban approach with the SCRUM approach. The latter, like Kanban, is an Agile project management framework developed to help teams manage their workload and structure their work.

The SCRUM team consisted of three members, each with a specific role and function: the product owner (Dieuwe), a SCRUM master (Fabian), and the developer (Wim). The product owner defined the vision for the project, using information from the product users, and ensured all players understood the tasks and subsequent time requirement for task completion. The SCRUM master, then planned the sprints, organized stand-up meetings, and reviewed sprint outcomes. The developer developed the tasks, performed the tasks, and moved task status to the SCRUM board, and uploaded the projects code to Gitlab.

To incorporate the SCRUM approach into the workflow, sprints were decided to last 1 week each, with sprint reviews scheduled during the weekly meetings. Tasks were given weights to account for the number of hours needed to complete, with a maximum of six hours. Tasks taking longer to complete were divided into smaller tasks. As illustrated by the burn up and burn down charts of a sprint (Figure 88), one of the benefits of moving to a SCRUM approach was the added value of enabling the team to gauge the progress of work and assessing the on-time completion of sprints.

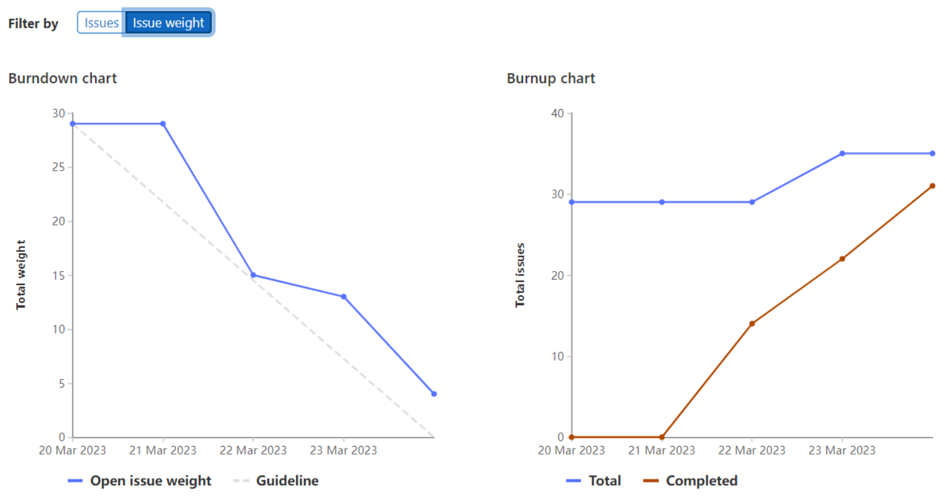


Figure 8 Screen shot of Burn Up and Burn Down chart of a Sprint

3 Quality Assurance

For this project, all code was written in Python. To ensure uniformity to code standards, PEP8 and YAPF guidelines were followed. Rules followed include line length, spacing rules, naming objects, and laying out codes. Collectively they ensured the compatibility and consistency of the code written and warranted the readability and overall understanding of the Python code.

A number of tests were run to determine the functionality and quality of the codes written. They included smoke testing, regression testing, and post-training testing. Smoke testing was performed early in the development of the codes to check the stability of the software written and to find major defects in the core of the software. For this project.

Besides smoke testing, the developed software was subjected to regression testing. Software regression is when a feature that previously worked ceases to work after a particular change is deployed within the software like source code changes, system updates, and so forth (Regression Testing, 2023). In this project, regression testing was performed after tasks were incorporated and the functionality of the system needed to be re-established. Tests targeted a deeper level in scope than smoke testing, meaning that all modules and methods were tested. The purpose of the tests was to identify if the changes had caused new bugs or had harmed the software's functionality.

Finally, software training assessment tests were performed. In this project, software assessment tests were performed after the software received training in identifying MTi sensor DM codes and their success could be compared to handheld scanning devices.

Post-training assessment tests were administered at the very end of Object Detection Model training and all training "modules" had been completed. The model was tested against a test data set, where the number of sensors and the position of sensor DM codes was known. The Model's output was then checked against the predetermined data set to assess not only the success of the Model but more importantly the effectiveness of the module's training providing valuable insight about the capabilities of Computer Vision for automated sensor code registration.

4 Research

Five research questions were generated to address the problems identified with barcode detection and registration. The first three deal with developing a viable Computer Vision system to automatically scan and register MTi-1 sensors. The last two then target the implementation of the developed CV system so machine operators can effectively and efficiently use the system. Collectively, they assess the practicality of implementing a full-scale automated system for MTi-1 product registration and tracking.

Since the creation of a Computer Vision System was requested by Movella, the main research question of the internship will be: **Is it feasible to create a Computer Vision system to register DM sensor codes automatically?** To answer this, it is important to break it down into sections. First Computer Vision will be using a camera to see the sensors and Matrix Codes, which will then need to be processed and registered. So the first sub question will be, **how can CV be used to identify and register sensors?** Once approach is identified, tools to support it will also need to be selected. The second sub question therefore will be: **Which tools are needed, and what physical setup is required, to create a CV system that can successfully read DM barcodes?**

In order to address the operators' problems, the system will need to effectively communicate errors it may encounter when reading the tray of sensors. **How can sensor detection errors be effectively communicated and/or visually displayed to end users?** To ensure the product is created to be easily used by the production operators, maintaining communication, and receiving their input during the creation of the system will be import. **How can machine operator insights and feedback be used to improve the operation of the system?**

4.1 Question: How can CV be used to identify and register sensors?

To find out if CV can be used to replace the manual scanning of each MTi-1 DM code, a literature search of Computer Vision was conducted to identify approaches which can be implemented to create a way to identify DM barcodes within an image.

Template Matching. According to (OpenCV Template Matching, 2023), template matching is a technique which searches through a given image to identify pixels that match the template image. By returning an average of the local pixels which do match, and by displaying them in the form of a pixel map, where the brighter the pixel area, the closer the match to the template. While the approach appeared to be quick and seemed useful when scanning object images that follow specific patterns, the main drawback for using Template Matching is that the technique compares pixel values for very specific patterns and thus works best when the template and object are very similar. Given the fact that DM codes have different code patterns, the use of Template Matching in this project could lead to detection issues as the template will likely differ from the DM codes being detected.

Machine Learning using a Haar Cascade. Also known as Cascade Classifier, this approach searches through an image looking for "Haar Like Features" (Figure 99) which

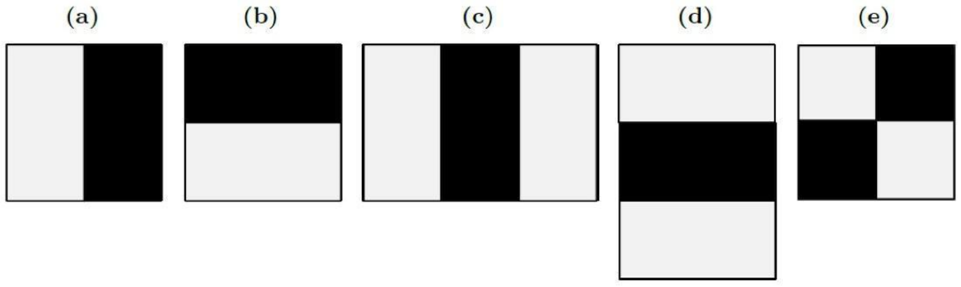


Figure 9 Haar like features Sourced from OpenCV Haar Cascades

create a sum of pixel values on the white side and the black side and then compare the two. By stacking a lot of features on top of each other, known as a cascade, detection of a complex object is obtainable. According to Cascade Classifier (OpenCV Cascade Clasifier, 2023), to find the features that will identify an object, the Haar cascade uses example data known as positive images (images which contain the object) and negative images (images that do not contain the object). In the positive images, the object to be detected is outlined so the training algorithm knows if it is correct or not during training. Training is done by going through multiple rounds, where it uses the Haar-like features to guess where the object is (OpenCV Cascade Clasifier, 2023).The closer to the expected result, the more that feature is considered important when considering if it is an object or not. Features that always guess wrong are therefore given a negative weight, and the cascade will do the opposite of what the feature outputs, which is used to eliminate potential matches earlier in the cascade which improves its efficiency.

Neural Network. Similar to Machine Learning, this technique requires training. Training is needed to generate nodes, which look for small patterns and then assign a weight to them (Habibi, 2017). The greater its weight, the more important that pattern is to identify an object. These nodes are created in different layers of the model and combine the weights of nodes of previous layers to determine if an image contains the object (Fig. xx). While the technique requires longer training than the Haar Cascade, Neural Networks can be flexible in identifying objects that aren't always the same and are less likely to be overtrained by the training data.

In particular for computer vision tasks convolutional neural networks have been successfully for a wide range of applications exceeding even human capabilities. The convolutional layers in these networks allow to recognize patterns from small lines and edges which are than combined in deeper convolutional layers to higher level features which allow a precise detection.

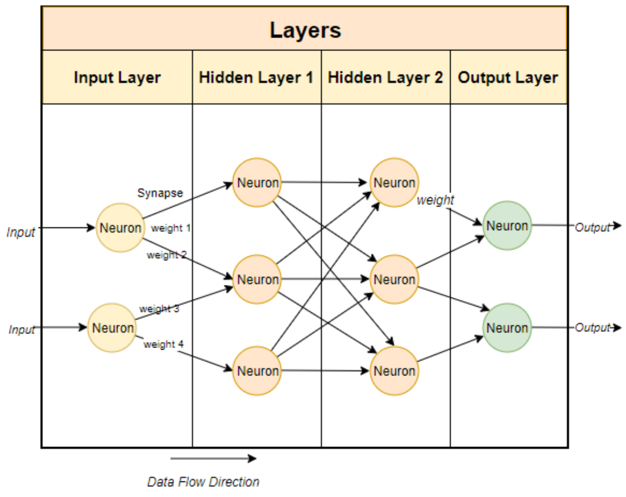


Figure 10 Neural Network

Comparing the two approaches, it would seem that Neural Networks are very powerful techniques that can be successfully used to accurately detect objects which they are trained to detect. However, the technique is very complex, takes a lot of time to train and their performance strongly depends on the amount of training data available. On the other hand, the Haar Cascade approach is less complex and takes less time, while still able to generate an accurate Object Detection Model (ODM). For the latter, a thing to watch out for is over fitting. This can occur when the training data is limited, and the ODM begins to filter out objects that do not perfectly match the training data. Resulting in the same draw backs as Template Matching. To avoid this from happening to the Haar cascade, the number of training rounds should be limited or, conversely the variance in the training data should be increased.

Given the time limitation for this project, it was decided to apply the traditional machine learning approach with the Haar Cascade to generate the Object Detection Model for reading DM codes on MTi sensors. While the approach is certainly not as adaptive as a neural network, it is easier and quicker to train, which will be important since it is likely there will be multiple iterations of the model while trying to figure out the best training approach for the detection model.

This project will be using the traditional machine learning approach with the Haar Cascade. While it's not as adaptive as a neural network, it is easier and quicker to train, which will be important since it is likely there will be multiple iterations of the model while trying to figure out the best training approach for the detection model.

4.2 Question: Which tools are needed, and what physical setup is required, to create a CV system that can successfully read DM barcodes?

Having identified an approach for testing the feasibility of using CV in automatically registering MTi sensor DM codes, the system's components needed to be identified and selected. A cursory review of the literature revealed the need for a reliable camera (Vandendorpe, 2021), a stable camera mounting platform (Norton, 2023), a flexible object detection model (Nanos, 2023),and an easy to use Graphic User Interface (GUI) (Pedamkar, 2023)The following paragraphs describe and discuss the various components identified for further testing and selection in the final product.

To best identify what kind of camera will be needed to read the DM codes on each of the sensors, a way to read the codes should be determined. To find answers, a literature search was conducted to locate a library containing a program tool that can be installed and used to read DM codes from images. Once a library is found, a number of cameras Movella already has can be tested. The cameras available for the project are: Logitech C270, RealSense D415, Chameleon 3 and Logitech C920. Shown below.

4.2.1 Image Capture

A large number of cameras are currently available to capture barcode images. The most prominent are digital cameras which can be linked to software which pre-processes the image to prepare it for further analysis where the image is ultimately decoded and read. In this project, the following digital cameras were used to acquire the MTi sensor DM codes: Logitech C270, RealSense D415, Chameleon 3 and Logitech 920 . All were part of the Movella inventory.

Figure 12 Logitech C270

Figure 11 RealSense D415

Figure 14 Chameleon 3

Figure 13 Logitech C920

The testing of these cameras will be carried out during the implementation phase and covered in more detail in section 6.1 where the cameras will be tested to determine the specific requirements for generating readable images of DM codes.

4.2.2 Camera Platform

After free-shooting a number of images it became clear that long exposure times were needed to acquire multiple images. As long exposure times typically lead to camera shake and subsequent blurred images, a physical structure was needed to stabilize the camera so high quality images needed to read the sensor's DM codes could be obtained.

It became also clear that, given the CV system's requirement to capture 20 or 100 images simultaneously, the use of a tripod to take overhead shots of a tray filled with sensors would not do. A frame needed to be constructed which was rigid, yet portable to facilitate experimentation with camera angles, positions, and light conditions.

It was decided to use aluminium for the construction of the actual framework as it was light, yet strong, and was easy to build. For this project, a stable H-shape stand was built by connecting two equal length "T-slot" bars with a single, aluminium crossbar. On top of the crossbar was a 3D printed camera mount fastened to the crossbar by a bolt connected through the t-slot of the aluminum bars (Fig. xx). As the crossbar was attached to the T-slot bars by simple corner joint fasteners, the height of the crossbar and subsequently the camera platform could be adjusted.

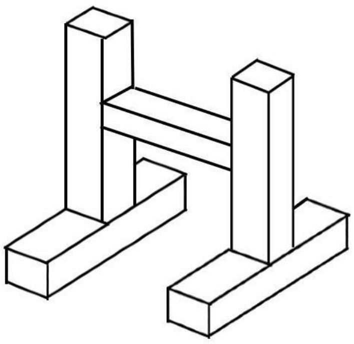


Figure 15 Diagram of the Assembled Prototype Structure

4.2.3 Image Detection

After acquiring an image, the digital camera sends it to the software to prepare the image for further analysis. This stage usually converts the image to grayscale and applies various filters to reduce image noise and enhance barcode edges. Binarization is typically performed next to ensure that black and white pixels remain in the image.

The decoding process consists of two steps: barcode location detection and barcode decoding. During the first step, software recognizes and extracts the barcode part from the complex image acquired by the camera. This step enables the system to identify the location of the barcode without the need for prior knowledge of the location. It also means that the model can locate multiple barcodes at once.

The tool used to detect the sensor’s barcode location is called Object Detection. Object Detection allows computer systems to “see” their environments by detecting objects in visual images or videos (Boesch, 2023). Methods for object detection typically fall in neural network-based or non-neural approaches. As it was decided (Section 5.1) to use a classical machine learning technique with the Haar Cascade was chosen for the detection of MTi sensor DM codes.

4.2.4 Image Processing

As indicated above, an image’s decoding process consists of two steps: barcode location detection (described in section 5.2.3), and barcode decoding. While several devices exist to capture DM codes, a few libraries are available to read and interpret a DM code. A literature search revealed that there is a library called pylibdmtx (Python Library for Data Matrices) which can scan captured images and not only return an array of data containing the data read from the sensor, but also the coordinates on the image where the code was found (Laghton, 2019). Another potential solution to reading DM codes is within the OpenCV Frame. A class capable of reading two dimensional barcodes called

QRCodeDetector (OpenCV QR Code Detector, 2023) Due to this class being part of a tool that was already decided to be used, it was worth looking into if it would work with DM codes. However, after testing, the class only works for reading QR Codes. So it was decided to go with the pylibdmtx library since it could read DM codes and was compatible with Python.

A drawback on the use of the Pylibdmtx library is that the higher the resolution of the image, the longer it takes the library to search for the codes. When a partial DM code and another DM is in the same image, the position of the rectangle (used to draw bounding boxes around detected objects) can be placed onto the unreadable code thereby overriding the rectangle of the readable code. This not only mislabels the two codes but also causes confusion as to which data matrix code was actually readable.

4.3 Question: What data is needed to train the detection model to handle non-conform situations?

Machine learning (ML) is a branch of artificial intelligence (AI) that focuses on the use of statistical methods to develop algorithms which are trained to make classifications or predictions(reference). For this project, ML using a Haar algorithm capable of detecting objects in images, regardless of their location and scale in an image is considered. The reason for selecting the framework is discussed in Section 5.1.

Like all ML algorithms, the ML object detection framework using Haar Cascade features required a large amount of data to make accurate predictions. To train the algorithm, a large and representative set of sample data needs to be collected. For this project, categories of training data included: different lighting conditions, distance from the tray, sensors not laying flat on the tray, rotated sensors and different camera settings and camera resolutions. To ensure the model could detect DM codes in non-conform situations, edge case data (i.e., extreme data) were incorporated. This to added variance to the training data, with the assumption that the broader and more varied the training data is, the less likely the model would be overtrained.

To develop such an automated CV system, the use of machine learning (ML) algorithms to identify DM barcodes will be investigated using OpenCV. The data from a set of ML training examples with different light settings taken with different cameras, some without DM codes, will then be used to enable the ML algorithm to identify the location of the DM code on the sensor and the object detection model to detect the DM codes in each image.

Using the OpenCV framework, there is a cascade training tool called opencv_traincascade.exe, this executable file is what runs the training algorithm. The file is run from the command line and takes a number of different training parameters that will ultimately affect the outcome of the trained model. These parameters will be explored further in the implementation phase. For now, the general inputs for the file consist of folders with positive and negative examples. The negative examples are images that do not contain the sensors with ID labels. While the positive examples are images that do contain the objects the model will be learning to detect (OpenCV Cascade Classifier Training, 2023). Along with the positive examples a text file will be provided to the system, this file lists the location of the image, along with the number of objects, with their position, width, and height. This text file tells the defines to the algorithm where the DM codes are, and what their bounding box is. During training, the model will guess what is and isn't a DM code, which the trainer will confirm whether or not the model is correct and make adjustments to the model from there.

An important part of the training data lies in the quality of the negative examples. According to the information found online, just any images without the objects to be detected will not be useful when defining what the object to be detected is not (Ben, 2023). A picture of a mountain for example may tell the model that the sensor is not a mountain, but when trying to scan an actual tray of sensors, there will never be a mountain. A better approach is to provide negative examples the model will likely see in the production environment, the tray for example provides better context. This tells the model the tray in which the sensors are placed on, is not actually a part of the sensor label itself.

Figure 16 Example of Bad Negative Training Data

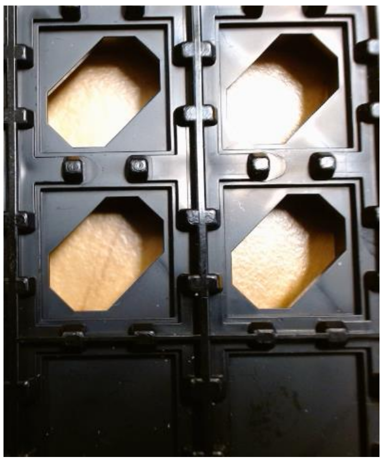


Figure 17 Example of Good Negative Training Data

Now that quality negative examples have been defined, it is time to look into what will be needed for the positive examples. When it comes to the positive examples, a high volume of the object to be detected is an important factor. The larger the data set, the more chances the model will get during training to guess where the object is. For the general detection, good resolution, lighting, and consistent distance will be important. The idea behind this category of positive examples is to show the model that even though the black and white pixels aren't always in the same place, the white label on the sensor with a square grid is what it should be looking for. If the training data is limited, it could be the model is trained to only recognize certain codes, which is what should be avoided. The figure below shows a high-resolution image of a sensor label which will be used as training data for the model.

In training the Haar algorithm, it is important to use both positive and negative images where positive images represent situations where the to be detected object is present, and negative images where the to be detected image is not. The trained algorithm was then used to detect objects in images from test data. The figure below shows an example of a positive image that was used as training data for the ML model's algorithm.

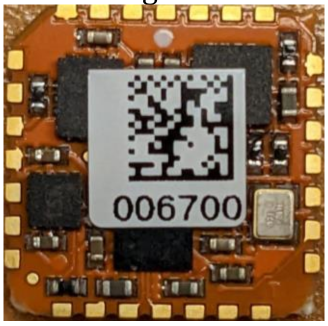


Figure 18 Example of Good Positive Training Data

The previous paragraph mentions high volume of clear labels as a category of the training data. It is called a category, because the model will not always be shown DM codes in the most perfect conditions. Time of day and weather could change the lighting of the images being captured, along with other unexpected variances. To help ensure the model can still detect DM codes in non-conform situations, edge case data needs to be taken into consideration. Other categories of training data include: different lighting conditions, distance from the tray, Sensors not laying flat on the tray or rotated slightly, even different resolutions from the cameras that will not be used in the final product can add variance to the training data. Which in the end will broaden what the model is considering when determining if an object is a DM code. The broader the training data the less likely the model is to be overtrained on the data set.

4.4 Question: How can sensor detection errors be effectively communicated and/or visually displayed to end users?

Once the CV system is operational and functions as expected, it is still possible that the detection of a sensor is wrong. It is therefore important to bring the faulty detection to the operator's attention. When a sensor is detected incorrectly or missed entirely, the system will not be able to read the DM code. In this case the operator will need to physically adjust something on the tray. Whether it be resituating the sensors or replacing a sensor with another. In either case, the operator will need to know what the issue is and which sensor on the tray is affected.

In section 5.2, The option of using ML to identify where sensors are and crop them into separate sub images to increase the efficiency of the library was mentioned. The sub images would then only have one Matrix Code, it is then known exactly which code is unreadable because the return value will be for that specific Matrix Code, and as the detection model will know it's position, it is then also possible to annotate which sensor in the image was not readable using the OpenCV library for python. In this library a rectangle can be drawn on an image at specified coordinates. Along with having custom colours for the lines of the rectangles. By indicating to the user what each colour means, the status of individual sensors can then be visually shown on trays of 20 and even 100 sensors without cluttering the screen with status text. While also showing the user where on the tray the problem sensors are.

4.5 Question: How can machine operator insights and feedback be used to improve the operation of the system?

The best way to find out how well a system is operating is to ask the end user. Several tools are available to solicit end user insight and feedback. They include personal interviews, surveys and questionnaires. For this project, conversational feedback was chosen as the preferred tool to solicit end-user feedback and insights as it is casual, fun, and non-invasive (Haije, 2023). Specific questions target how users experience the current process and what they wish the system can do to ease their job.

It was found that the operator's main concern is the accidental processing of sensors if one or more of the sensors aren't registered. This is likely due to issues with the current use of barcode scanners, which could be caused lighting glare and subsequent improper scanning or missed scanning of a DM code.

To address this concern, the CV system's GUI has a built-in screen with clear indicators of when all sensors have been registered and also when issues are encountered with the processing and registration of sensor DM codes. In case of the latter, the flow of sensors is stopped immediately and does not proceed until all sensors on the tray have been processed and registered.

Another feedback received from machine operators is notification on whether an issue with a sensor's code reading can be easily fixed (e.g. repositioned) or if the sensor needs to be replaced by another one. To address this concern, a wireframe diagram was developed (Figure 1919) which facilitates the detection and mitigation of common issues.

In its current setup the information portrayed is rather limited. However, the information can be expanded by adding other detection or registration issues. Such information might prove helpful in separating minor issues which can be easily fixed by machine operators in the form of repositioning sensors from major issues which require the replacement of sensors and the subsequent re-reading of sensor codes.

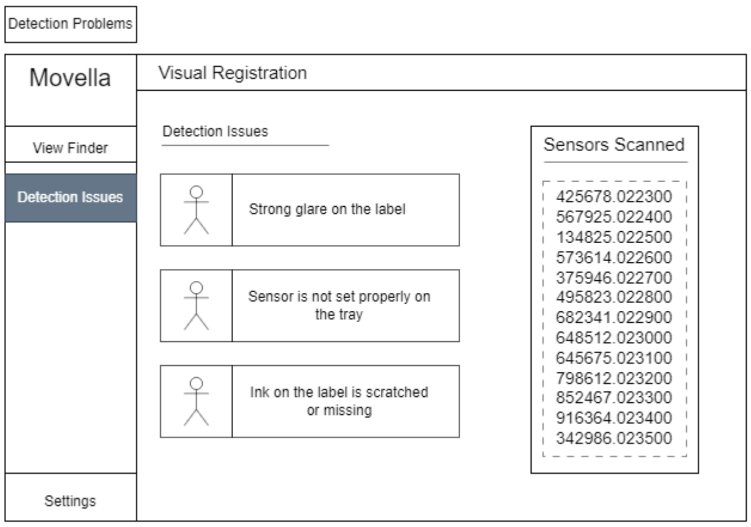


Figure 19 Diagram of the Issue Page to Help Identify Known Detection Issues

5 CV Component Testing

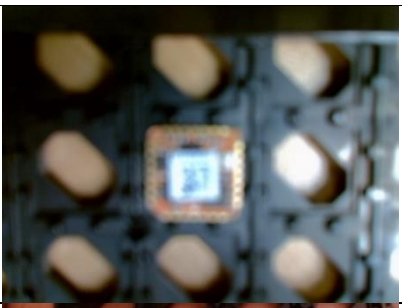
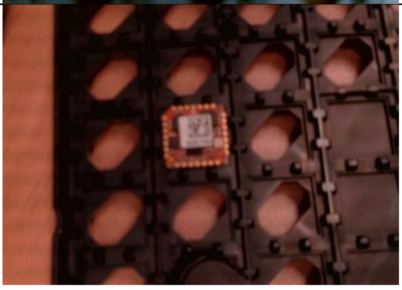
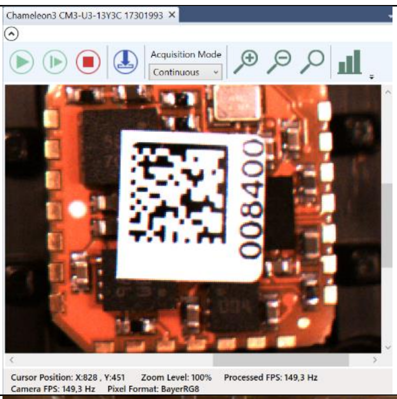
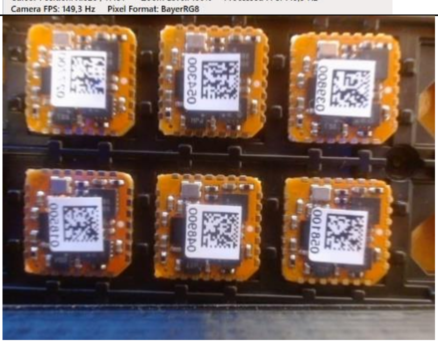
Using the outcome from the research questions, a plan of approach was developed to generate a conceptual prototype for computer vision registration of Movella MTi DM codes. This chapter reports on the various system components making up the conceptual prototype. The first two sections describe the cameras considered and assessed for possible system integration and the construction of a physical platform to ensure the quality of the images taken. The following two sections then discuss the object detection model tests, and the training of the Haar Cascade to enable barcode detection and barcode decoding, respectively. The last two sections then discuss the graphic user interface (GUI) to help machine operators identify and (potentially) fix issues with the CV system, and the languages and frameworks used in the project.

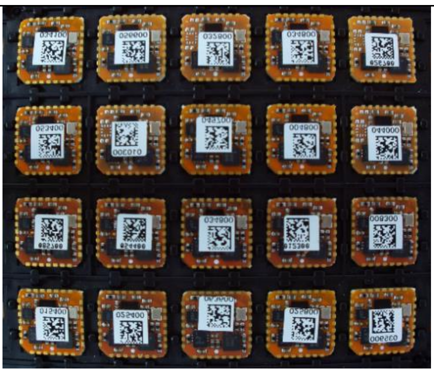
5.1 Build Approach

5.1.1 Selecting a suitable camera.

As the main focus of this project was to identify and read MTi sensor DM codes, selecting the right camera to meet the project's requirements was of utmost importance. To achieve this goal, a number of cameras were tested (Section 4.2). The following is a list of the various cameras tested for this project, along with an outcome of the tests.

Camera	Comparison	Result

Logitech C270	While the camera was able to read larger DM codes, it failed to read the smaller DM codes on MTi-1 sensors. It is believed that the camera’s low resolution and fixed-focal length were responsible for this deficiency.	
RealSense D415	With a resolution of 1280x720 pixel IR camera and a 1920x1080 pixel RGB camera, the camera was designed to measure object depth from as little as 16 cm distance. While the camera was easy to operate it failed to capture the small size of the sensor's Data Matrix code. The camera’s fixed focal length could be the cause for this.	
Chameleon 3 USB3	With an attachable macro lens, the camera provides easily readable images which can be put into focus at close range. As the camera shoots one image at the time, the sensors will have to be loaded into a reel for the camera to work. As the project targets the registration of multiple sensor codes simultaneously, the camera was dropped from further consideration.	 Acquisition Mode: Continuous Cursor Position: X:828 , Y:451 Zoom Level: 100% Processed FPS: 149,3 Hz Camera FPS: 149,3 Hz Pixel Format: BayerRG8
Logitech C920	The camera was an upgraded version of the first camera tested. With a higher resolution, and autofocus lens, the camera was able to clearly read multiple images at once. However, with the lighting glare off the DM labels the environmental conditions and camera setup required constant adjustment to get clear images.	

Pi Camera HQ	After assessing the capabilities of the four cameras and determining their shortcomings, a Raspberry Pi HQ camera was purchased. The camera has a resolution of 4056x3060 pixels and an adjustable focus of 12.2 to 22.4 mm. Tests revealed the clear images and subsequent detection, identification and readability of all 20 DM codes on a full tray of sensors.	

After aassessing the camera's capabilities against the pylibdmtx library, it was determined that the camera's resolution played an important role in obtaining clear DM code images. The best results were obtained with the Raspberry Pi HQ camera, as it not only had the highest resolution but also was able to obtain 20 readable DM code images of a tray of 20 sensors of the five cameras tested. For this project, it was the camera of choice.

5.1.2 Camera Platform Construction

Early on in the process it became clear that taking pictures while experimenting with light, camera angles, focus, depth of field, etc., was a tedious task. The most challenging of all turned out to be securing the stability of the camera while taking the shots. As only tripods were available to achieve the task of stabilizing the camera, and none were fitted to take overhead shots, the decision was made to build a customized structure.

To facilitate experimentation with camera angles, positions and light conditions, a portable structure was constructed. The platform consisted of an aluminium stand made of horizontal bars in the form of two "Ts" connected by a horizontal crossbar which could be raised to a maximum height of 38 cm and lowered to a height of 3cm to obtain the ideal image capturing setting.

Attached to the metal housing are 3D printed, "Thing verse" (Thingiverse, 2023) adapted camera mount to fit the Pi HQ camera selected from testing as well as a Raspberry Pi mount to interface with the camera. The two figures below (Figure 20) show the 3D mounts of the camera mount platform developed for this project.

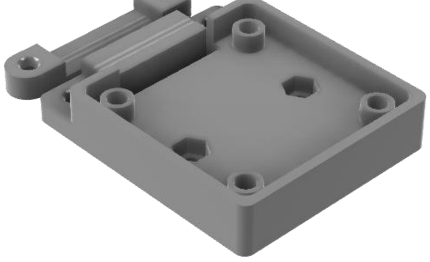


Figure 20 model of the Camera Mount

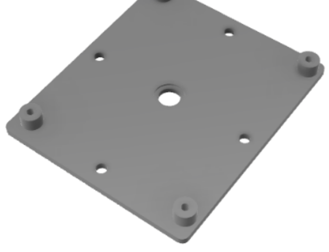
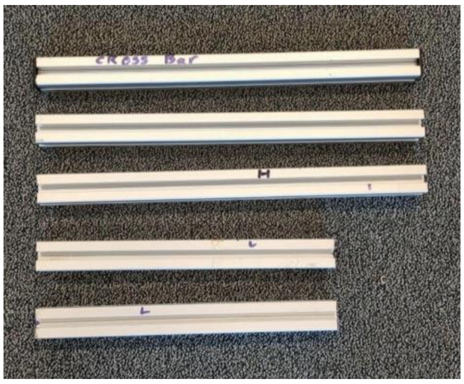


Figure 21 3D model of the Raspberry Pi Mount

The t-slots in the bars allow for the connection of right-angle brackets connected via bolts. This metal structure with tight fitting connectors made the outcome rigid and sturdy.



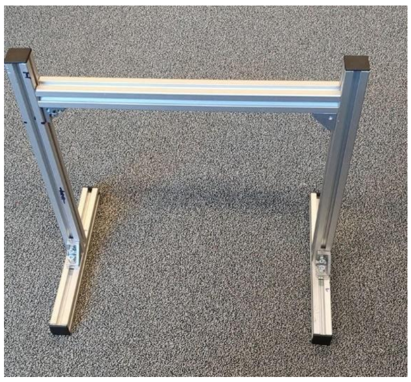


Figure 22 Unassembled Prototype Structure

Figure 23 Assembled Prototype Structure

Figure 22 shows the assembled camera platform for taking the shots needed to capture the image of all 20 sensors and their DM codes. This system provides a very stable base and the height of the cross bar is adjustable to allow for finding the best distance for the camera. Figure 24 below show the 3D printed mounts being mounted with the RPi Camera and microcontroller.

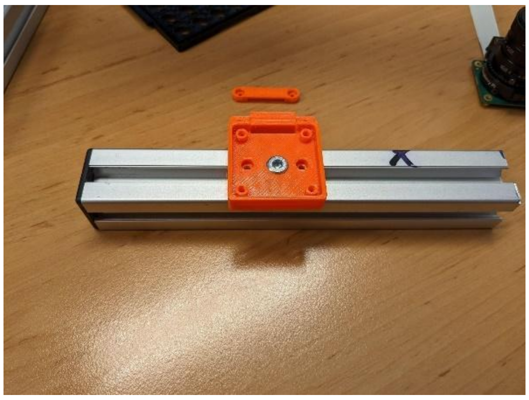


Figure 24 3D Printed Camera Mount

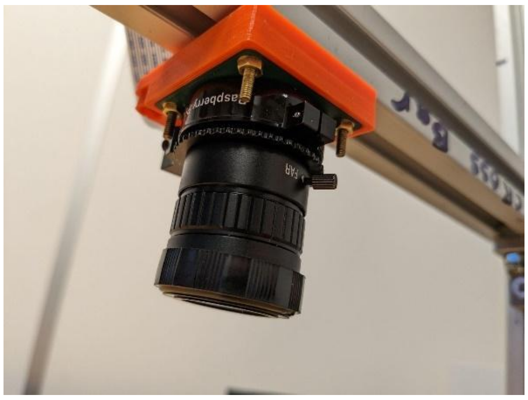


Figure 25 3D Printed Camera mount with the Pi HQ Camera

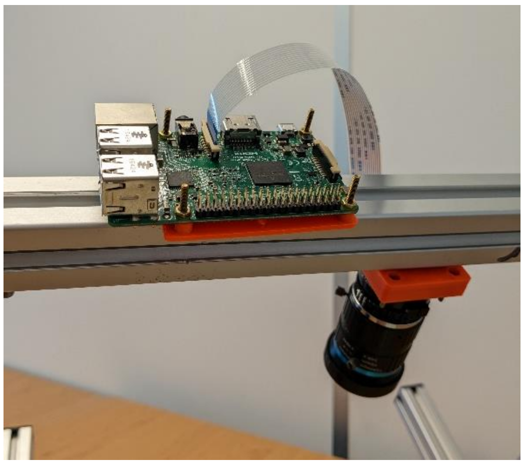


Figure 26 Raspberry Pi and Camera Mounted

5.2 Object Detection

In this project, Machine Learning (ML) algorithms were used to detect the barcode's location. To accomplish this task data was prepared to evaluate the efficiency of the barcode detection approach. By the end of the project a total of 1000 sensor images were used to create the DM code dataset.

After each iteration of training the Object Detection Model, the model went through a collection of testing data, where the position of the Data Matrix codes was already known. This data set was used to see how the model improved or regressed from the previous training attempt.

As the position of the Data Matrix codes in the test data was already known, the two bounding boxes could be compared and the Model's performance assessed using a technique called the Intersection of Union (IoU) (Mesquita, 2021). This approach takes the Area of Union, which is the total area of the two bounding boxes when merged (if they overlap) and divides it by the Area of Overlap.

By examining the degree of bounding box matchings, the precision of the ODM can be determined. The higher the percentage the better the match. If no overlap between the two bounding boxes can be found, the image is classified as False Negative, meaning that the model should have detected the image but for some reason did not. On the other hand, if a bounding box from the ODM is provided where there is no true bounding box, it is classified as a False Positive, meaning that the ODM should not have identified a DM code image but did.

In the case the two bounding boxes overlap, the ODM is thought to be correct. However, it still needs to be determined if the model was lucky and accidentally got close in detecting the DM code, or if it really detected it. In such cases finding the IoU might prove useful. If the IoU is higher than 60%, the ODM is close enough for the pylibdmtx library to read the code and classify the image as True Positive.

Figure 27 shows the performance results of 8 ODM's built in this project. The first three models were trained using a limited dataset (100 sensors). The next five models were built using image augmentation to increase the number of training data. This was achieved by rotating the barcode centre in increments of 90^0 (more information is covered in 5.2.6). The number of training samples used in the last five ODMs tested was therefore 500 vs 100 in the first three models.

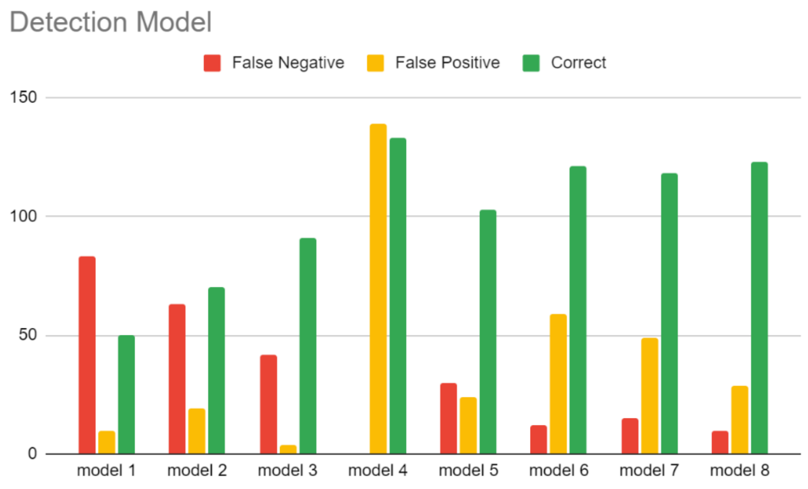


Figure 27 Graph of ODM performance after training

As the goal of object detection is to obtain as many True Positives, and as few False Positives and False Negatives as possible, it was found that Model 8 yielded the best results of the eight models tested. This model was subsequently selected for integration in Movella's conceptual CV system for automated sensor registration.

5.2.1 Image Processing

Detecting the DM code on a sensor is only the beginning in registering the code. Reading and interpreting the image is the other part. A literature search revealed that there is a library called pylibdmtx (Python Library for Data Matrices) which not only can scan captured images but also return an array of data containing the data read from the sensor as well as the coordinates on the image where the code was found (Pylibdmtx library, 2022) In this project, pylibdmtx was used for image processing.

In searching the site, it was discovered that the pylibdmtx library not only can read the data from a Matrix Code, but to a certain extent can also detect data matrix codes. This generated the question that if the Data Matrix Library can find a data matrix Code on its own, why use a Cascade Object Detector described before?

To test this, three example images were taken of trays containing 20 sensors each. The images were captured by the Pi HQ camera and stored locally on the computer. To ensure the test is reproducible and only takes into account how long the library takes to read all 20 of the sensors, the test program does not use an object detection model. Instead, the positions of the sensors were stored in an annotation text file.
The positions were then used by the program to identify where to crop the image. Cropping was done using an OpenCV method and returned a new image. The time an object detection model took, was not taken into consideration for this test as the goal was to find out if giving the library smaller images would reduce image processing time and justified the time it takes to train an ODM. Figure 28 shows the full sized image of 20 sensor codes, while Figure 29 depict the cropped versions of the left top sensor code and bottom right sensor code, respectively using the OpenCV method.

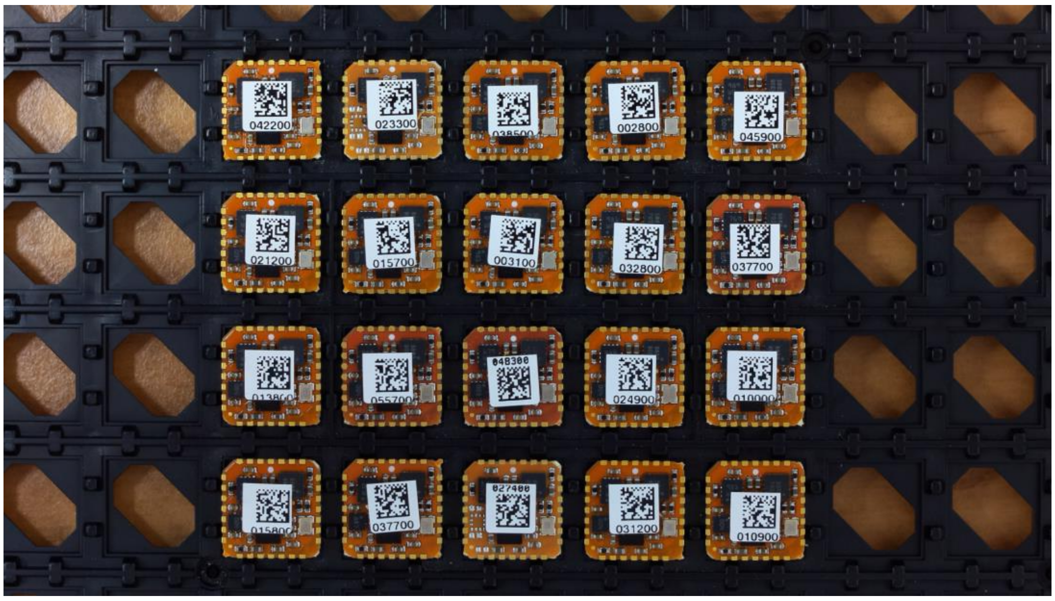


Figure 28 High Resolution Image of Tray with 20 Sensors

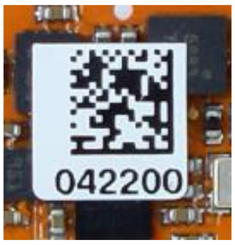


Figure 29 Cropped Image of Top Left Sensor

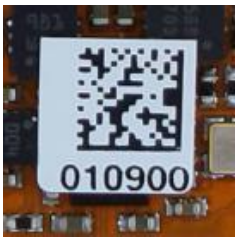


Figure 30 Cropped Image of Bottom Right sensor

Comparing the processing time of the same three uncropped and cropped images it was found that the uncropped (full-sized) Images took on average 29.8 seconds to run, whereas the array of cropped images took on average 5.4 seconds (a decrease of about 82%!). Stated differently, the time the library spends on DM matrix code image processing in cropped images is roughly 5 times faster than in uncropped images.

The reason for this is that the pylibdmtx library searches the entire image comparing pixel values from the top left to bottom right. So, when the image is cropped down to only contain the Data Matrix Code, the library does not need to search the image for a Data Matrix for very long, thereby reducing the run time. This suggests that the Cascade Object Detector should be used to locate the Labels, and return the image coordinates, on the image, and then use OpenCV to crop the image and the pylibdmtx library to read the image. (Code of reading the cropped images can be found in Appendix C.2 Matrix Reader)

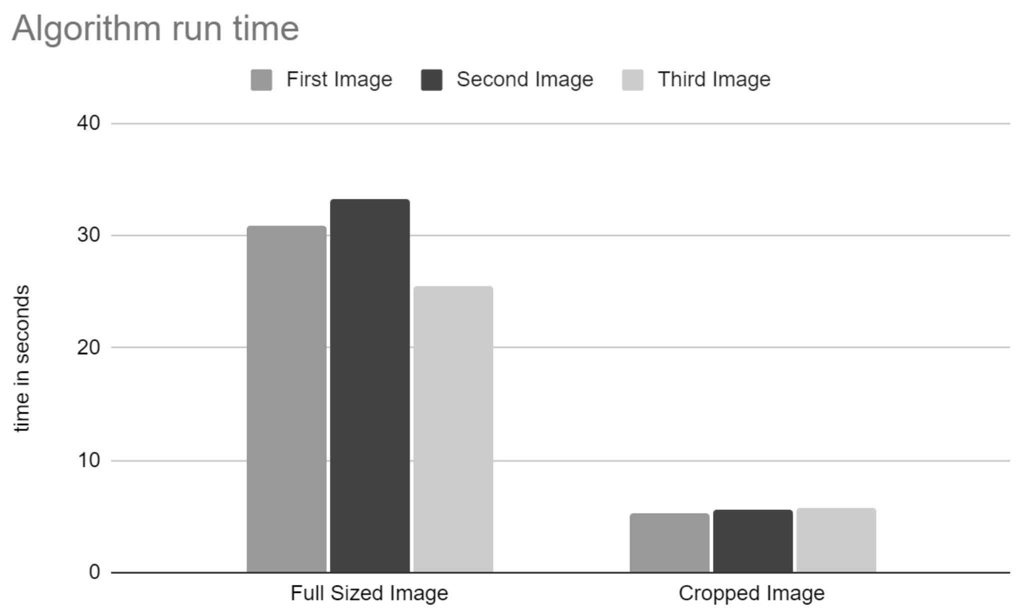


Figure 31 Graph Comparing Run Time of the Pylibdmtx Library

5.2.2 Capturing the Training data

In order to train the Cascade Object Detector collecting a wide range of data is important. The physical structure will allow for collecting high quality training data. However, one of the requirements of the project is for the algorithm to be robust and handle outlying situations. For the main use case of the detector high quality images are expected, however different lighting conditions or some amount of distortion, sensor misplacement, or even blurriness can be introduced through everyday use. To ensure the Detector can always detect sensors even in unexpected conditions it will be useful to have training data that isn’t in perfect condition.

This means that using the other cameras that were a potential match for the project outlined in section 5.1.15.1.1, should be used to gather a wide range of resolutions. Namely the Chameleon 3 and Logitech C920. As well as collecting data from different lighting conditions and distances, and even different angles.

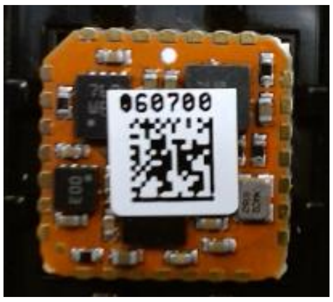


Figure 32 Consistent lighting

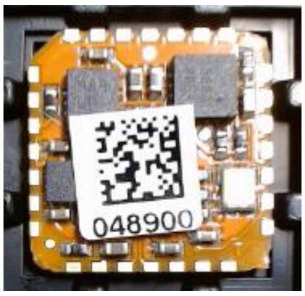


Figure 33 label alignment

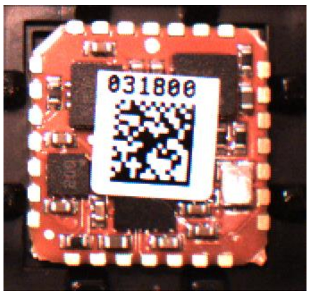


Figure 34 Chameleon 3 camera

The second phase of data collection was of the full trays after the physical structure had been created and the best training parameters had been found, see section 5.2.4 for more details. This was also collected in varying lighting conditions, and different distances since the structure can be adjusted to different heights. This section of data collection was all taken on the Raspberry Pi HQ mounted camera.

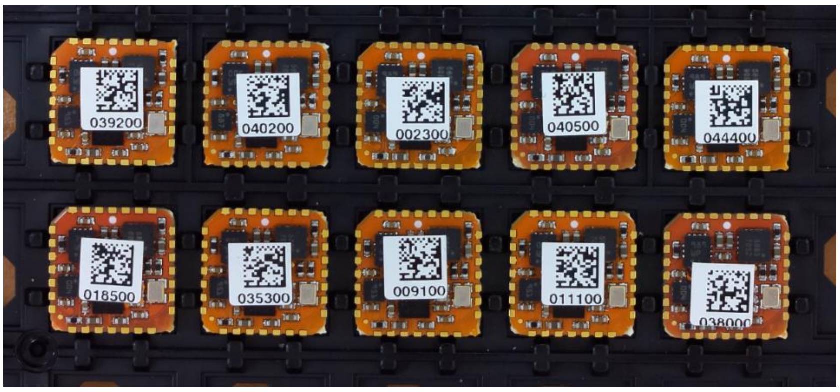
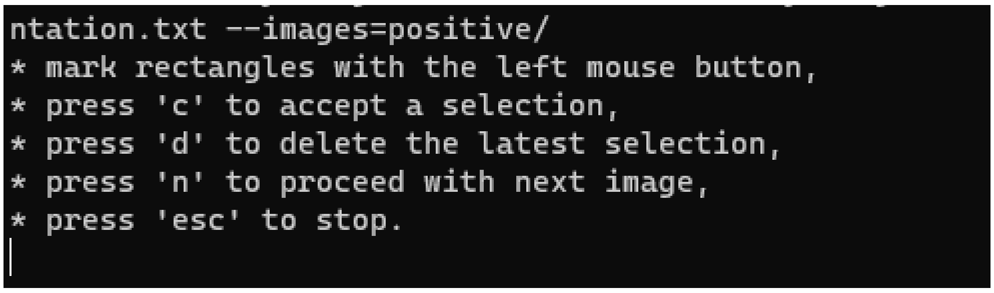


Figure 35 Pi HQ training data

5.2.3 Annotating the training data

The OpenCV annotation tool is a command line tool, which a user provides the name of an output text file (a text file which holds the data of the objects in the positive images) and the name of the folder containing the positive images (images containing Data Matrices). The annotation tool can be called in the command line by typing the following command:

opencv_annotation.exe –annotations=pos.txt –images=positive/



```
ntation.txt --images=positive/
* mark rectangles with the left mouse button,
* press 'c' to accept a selection,
* press 'd' to delete the latest selection,
* press 'n' to proceed with next image,
* press 'esc' to stop.
```

Figure 36 command line interface

The positive folder is then iterated through by the tool, where each image is displayed individually. For each image, the tool allows the user to select points on the image with the mouse cursor to create bounding boxes around the object and allows for multiple objects per image. When using the tool to select an object, the user clicks to the top left of an object, then selects the bottom right, which creates a red bounding box around the object. To confirm the bounding box is in the correct position, the tool has the user press 'C' on the keyboard which will turn the box green. If the box isn't correct, then clicking on a new position on the image will create a new red box. Once a box has been confirmed it will turn green and another box can be drawn. If it is determined after confirming a box, that it isn't where the user wanted it, they can press 'D' on the keyboard which will delete the previously confirmed box. When all objects in an image have been annotated, pressing the 'N' key will move to the next image within the positive folder. The following images show what this process looks like when annotating an image.

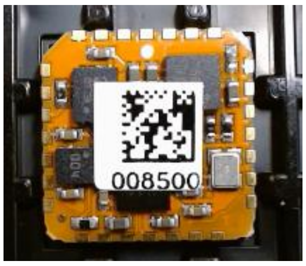


Figure 37 unannotated sensor

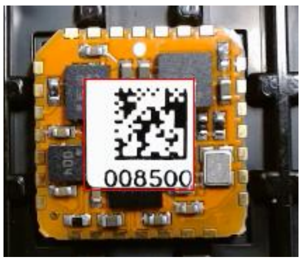


Figure 38 unconfirmed annotation

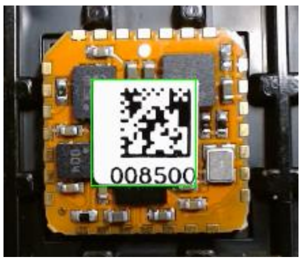


Figure 39 Confirmed annotation

After all images have been processed by the annotation tool, it will create a text file containing the file path to each image, along with how many objects per image, and the position & size of the bounding boxes.

Below you can see an example of the output annotation text file:

positive\sensor30.png 1 305 167 82 80
positive\sensor31.png 2 200 264 83 80 385 44 83 82
positive\sensor32.png 3 176 101 84 84 168 308 85 81 360 300 80 82

Each line of the file is associated with an image. The image path is the first part of the line, next the line indicates how many objects are in that image. The next two numbers refer to the top left coordinate of the object, and the following two numbers are the width and height of the bounding box. These four numbers (x, y, w, h) then repeat for as many objects that are in that image.

This text file is then used to make a vector file, which is the positive image reference to train the object detection model. OpenCV has another command line tool called create samples and takes the following inputs. The text file created by the annotation tool, the width and height of the window for training (usually 24 x 24 pixels (OpenCV Cascade

Classifier Training, 2023)), the number of objects in the annotation file, and lastly the name of the output vector file. The command is shown below.

```
Opencv_createsamples.exe -info pos.txt -w 24 -h 24 -num 1000 -vec pos.vec
```

The size of the window will need to match with the parameters given to the training tool and must be consistent with the vector file. The window is used as the starting point of the detection model when scanning an image and will need to be bigger than the smallest size the object can be detected. It is recommended to use a size of 24 x 24 as the larger the window is, the longer it will take the model to train. (training a cascade classifier, 2020) The vector file is used by the OpenCV training tool as the positive object input, which is covered in the next section 5.2.4.

5.2.4 Training the Object detection with different parameters

Training the detection models is done by the OpenCV train cascade tool. Like the two tools outlined in section 5.2.3, it is also executed from the command line. The parameters it takes to train the detection model are, the vector file generated from the last section 5.2.3, a txt file containing the file path to all negative images, the number of positive objects in the vector file, and the number of negative examples (these are generated from random cut outs of the negative images), the number of stages, and the detection window size. The window size must be the same size as the vector file, in the case of this project that is 24 x 24. The parameters listed above are all needed to execute the tool, but there are more possible parameters that can be used.

5.2.5 Training the Haar Cascade Algorithm

There are several different parameters that can be selected for training the Haar Cascade. The main parameter that affects the training the most is the number of stages. This is the amount of times the Trainer will go through all the training data. At the end of a stage, the program looks at how the Object Detector performed. Images the Detector performed poorly on are more likely to be shown in the next stage, this is called AdaBoosting (Adaptive Boosting). The reason this parameter is so important is due to how it affects the outcome of the Detector. If there aren't enough stages, then the Detector will not have enough features to look for and will just be guessing. However, if there are too many stages, then the Detector can be overtrained on the data and will not accept objects that do not perfectly match its internal requirements.

One other parameter that is important to consider when training is the max false alarm rate, which makes the stage train until the detector is below the maximum rate of incorrect guesses. This parameter is important because without it, the stages do not train for long and will result in a detector that thinks everything is the object it's looking for, the result of this can be seen in **Error! Reference source not found.**40.

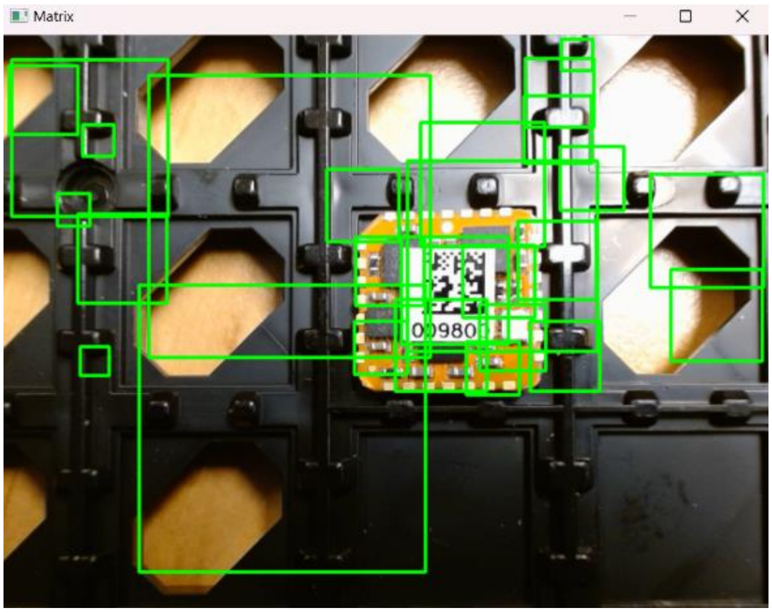


Figure 40 High False Positive Detection

The first attempt in training the Cascade Object Detector resulted in more False Positives than there were initially expected (See **Error! Reference source not found.**). This was due to not knowing about the MaxFalseAlarmRate parameter as mentioned in section 6.1.5. Once this parameter was set, the result was much clearer on where the DM code was.

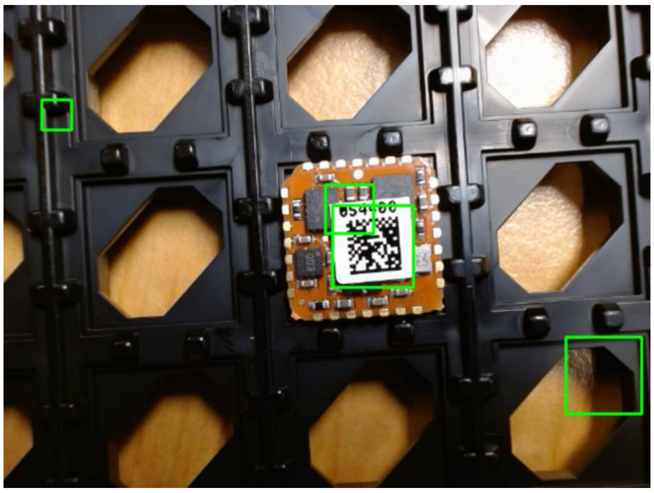


Figure 41 model 2

After each iteration of training the Object Detection Model, the model was used to go through a collection of testing data, where the position of the Matrix Codes was already known. This data set was used to see how the model improved or regressed from the previous training attempt.

The goal for the Detection Model is to get as little False Positives and False Negatives, while getting as many True Positives as possible, using the same training set. This is to reduce the amount of total training time, by finding the best parameter weights with the initial data set while it is quick to train a model, the final Detection Model can use these parameters on a larger set of training data where the resulting training time will take the longest.

One interesting observation of how the models reacted to the test data, is if the training was too strict then upside down labels were returned as a False Negative as seen in the left image below. Then if the training was allowed to be more flexible, the model would be able to detect sensors that were upside down but would be more prone to returning False Positives.

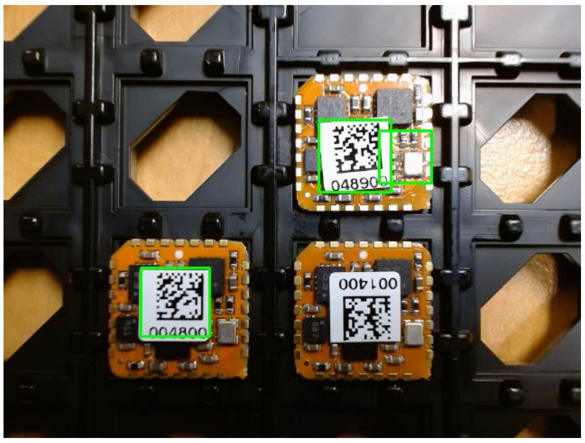


Figure 42 Model with Low False Alarm Rate (0.15)

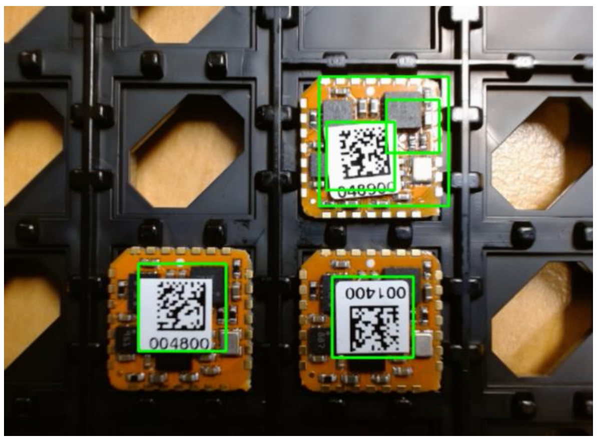


Figure 43 Model with a Medium False Alarm Rate (0.3)

Models that had a looser False Alarm Rate were able to correctly detect more DM codes overall, but also had a higher rate of False Positives. Adding data to the training set can help improve the False Negative rate, but lowering the rate of False Positives is done by lowering the False Alarm Rate which restricts the model. Since the models with a tighter False Alarm rate struggled on upside down labels. More training data with upside down labels should be added. This gives the model a better definition of how the labels can appear in images, while maintaining a low False Alarm Rate.

Another issue that was found with the original training data, is that bright or white squares caused by glare were identified as DM codes. This is likely caused by the Haar-like features Identifying a white square to be highly likely to be a DM code, if it has a white line to the left. In the image below an example of how the Haar-like features could mis-interpret the patterns and detect a code next to where an actual DM code is.

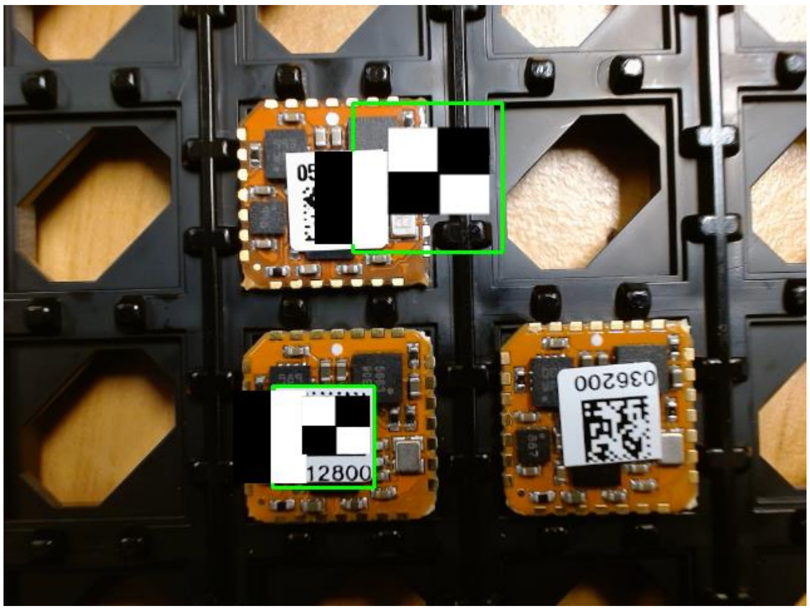


Figure 44 Detected Labels with Haar Like Features Overlayed

For this specific case, adding more examples of DM codes may not be sufficient. This is because the detection is finding features that are similar to how a DM code presents. To remove these types of False Positives, better and more specific negative training data can be introduced. By adding more images that are confusing to the model, the stronger it will get at determining what isn't a DM code. These negative images contain examples of brightly lit and shiny metal squares on the MTi sensors. Sensors with blank labels are also an addition to the negative data set, to better define to the model, that it should be looking for the DM code itself, and not necessarily the white label it is printed on.

5.2.6 Annotation Augmentation

Using the OpenCV annotation tool for the initial sets of training data was useful, however, once we decided more data was necessary to train the object detection model to be more accurate, it would become too time consuming to annotate all of the images by hand. Which presented the problem of how we can create annotated images automatically.

To solve this, predetermined bounding boxes are labelled on a select amount of images containing MTi sensors with a blank label. Then by generating random Matrix Codes and pasting them automatically onto the position of where the Matrix Code should be, it is possible to create a large amount of different codes, all pre-annotated and ready for training the Object detector.

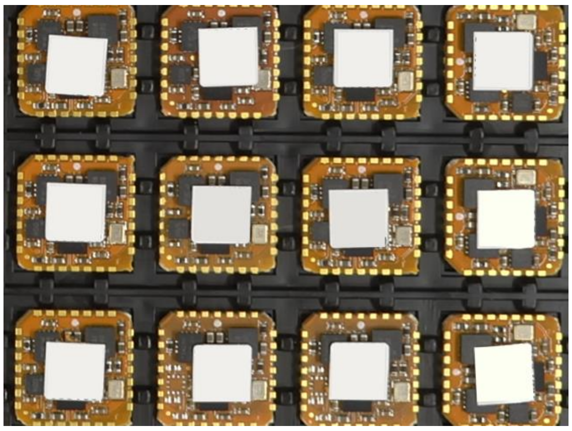


Figure 45 Blank label training data

5.2.7 Improving the accuracy of the model

In order to fix the issues discovered with the original data set a mix of Annotation Augmentation (generating pre-annotated Matrix Codes for the training data set) and adding negative images that addressed the white squares being detected as Matrix Codes. This is possible because the Adaboosting approach will show the negative images more if the Detection Model keeps identifying a white square as a code. The image below shows the results of attaining better training data with specific images to train against false positives.

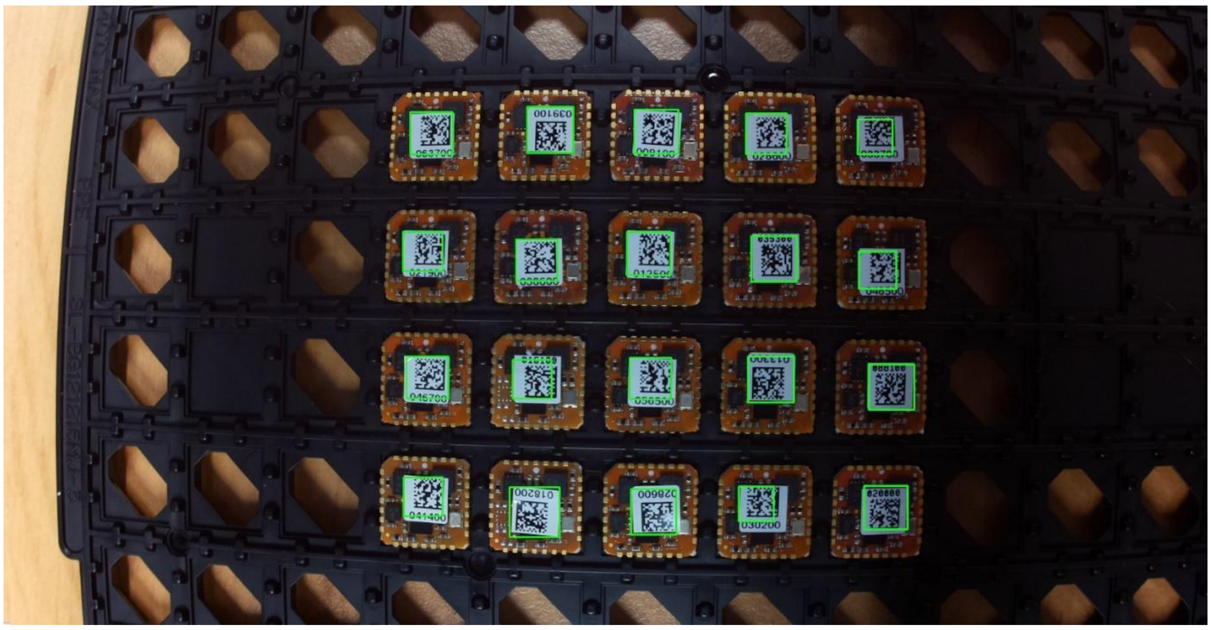


Figure 46 Image of the Detection Model Successfully Annotating a Tray of 20 Sensors

After making the adjustments to the Detection Model, the focus shifted to ensuring every part of the system worked as intended. The first part to address was when the system can not read a sensor, is it because of the detection model, or is it something to do with the library that reads the codes. If the library has inherent issues, a new library may be needed. Another part to consider is the image quality and if it affects the readability of the sensors. So, the next set was to work on getting a baseline established where the system can read the sensors 100% of the time if the conditions are perfect. The two following images show the result of the system on an image in what is considered the perfect conditions. i.e., proper lighting, image clarity, and distance from the tray.

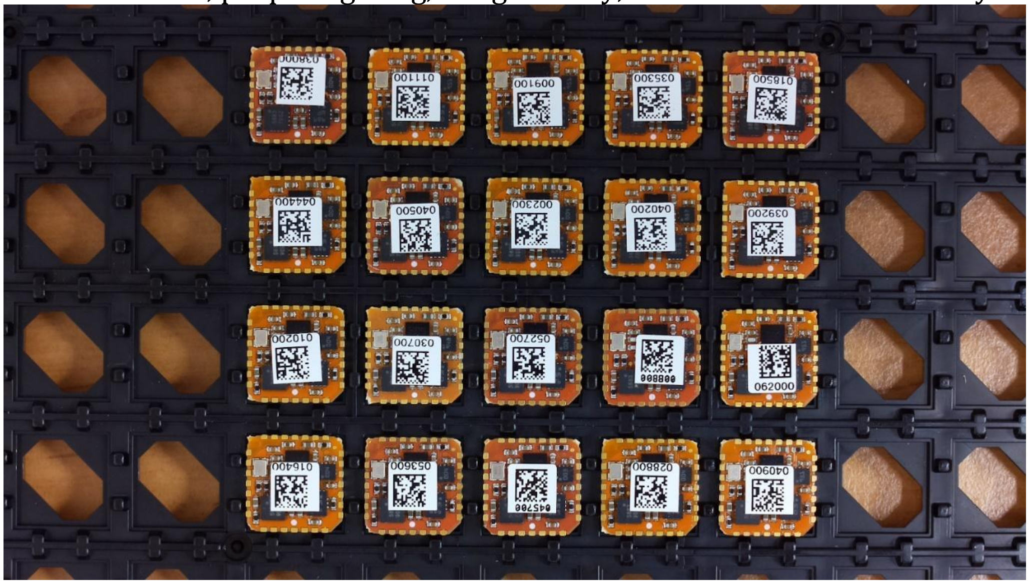


Figure 47 Tray in Perfect conditions

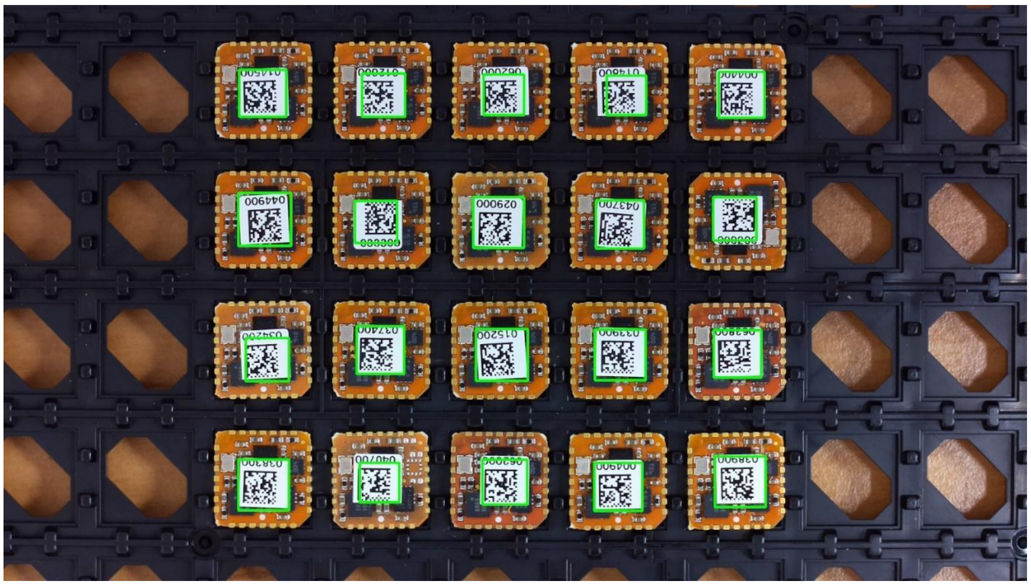


Figure 48 Tray of Perfect Conditions after Detection

When the conditions are perfect the library has no issues reading the sensors and the Detection Model can correctly Identify the DM codes almost all the time, with a few exceptions in false positives.

6 System Component Integration

Having selected and tested the conceptual CV system components to scan (individual) DM barcodes, the feasibility of a fully functioning CV system needed to be examined. To achieve this task, a conceptual CV system needed to be designed and the individual system components making up the conceptual product needed to be integrated, tested and evaluated. The following paragraphs describe the system design, the experimental setup for product demonstration, the tests performed to determine product functionality, and the evaluation of product demonstration and testing outcomes.

6.1 System Design

Before this chapter goes into the application of the decisions from 6.1, there is some context around the project that should be defined now that it is clear what steps will be taken to create the Visual Registration Product.

The production department currently uses a system called TestCal (test calibration) which uses the physical scanning gun to scan each sensor individually. This program is responsible for registering the sensors to the Movella database so that each order of sensors can be backtracked should a problem occur.

Each scanned sensor is entered into the TestCal program one at a time. Movella is currently working on a new system to replace the TestCal system, which will take in an entire list of sensors at once. To accommodate for this, the system returns a full list of the scanned sensors and can also export it to a temporary text file. As a temporary solution, a supporting program can take the text file of sensor IDs and enter each ID into the TestCal system individually until the new System is released.

The Context Diagram below shows how and where the system will be integrated into the production department and how it interacts with the TestCal system.

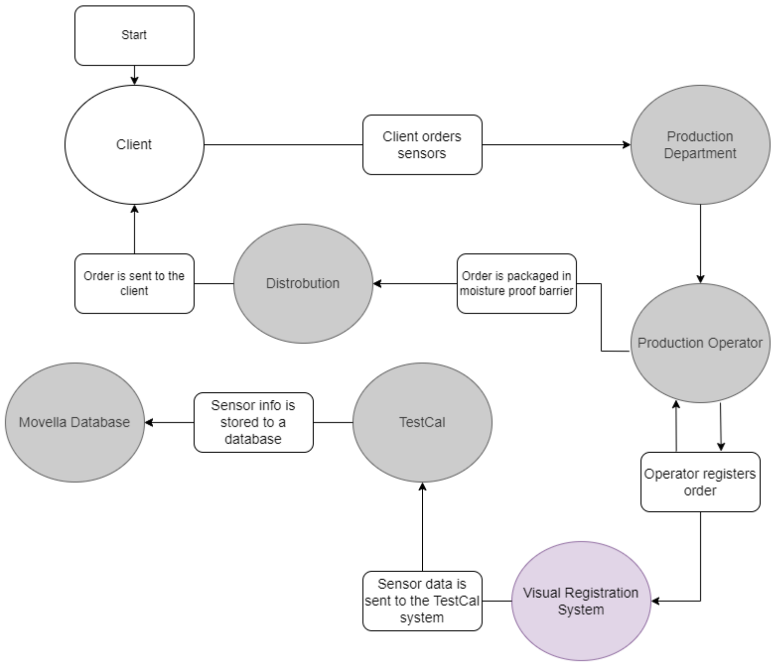

Start
Client
Client orders sensors
Production Department
Order is sent to the client
Distrobution
Order is packaged in moisture proof barrier
Production Operator
Movella Database
Sensor info is stored to a database
TestCal
Operator registers order
Sensor data is sent to the TestCal system
Visual Registration System

Figure 49 Context Diagram of the Visual Registration System within the Production Department

Now that it is clear how the system will be integrated into the production department, the system's architecture can be defined. The architecture will use a computer with a monitor that is available at each workstation in the Production Lab. The Visual Registration Program can be loaded directly onto the PC where the operator will use the GUI to interact with the program. On the workstation the physical structure with the Camera and Raspberry Pi attached will communicate with the PC over Movella's private network. A live feed from the camera will be sent to the PC and displayed in the GUI, when the operator is ready, they will begin the Detection and register the Codes. This architecture can be seen in the image below.

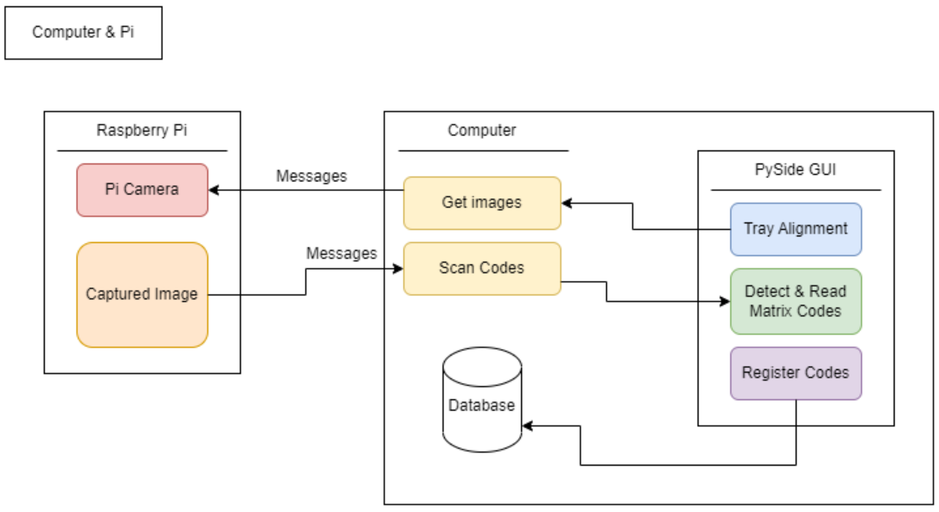


Figure 50 Visual Registration System Architecture

The Architecture will be implemented in this project. scalability to the system for the full implementation, the Object Detector can be hosted in an AWS cloud environment using Docker Containers. In this case the operator would be able to use any computer at any of the workstations to use the Visual Registration System so long as it had the physical Pi and Camera available. This would also make CI and CD easier to implement as only the cloud environment would need to be updated and not each individual PC in the Production Lab. The image below is the associated AWS cloud architecture for a scalable system.

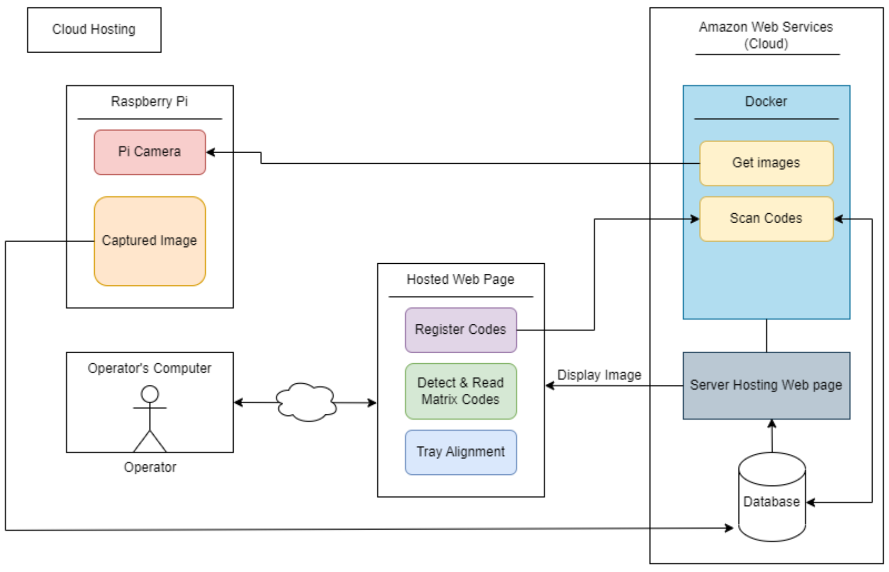


Figure 51 Cloud Solution Architecture

6.1.1 Data Storage

Seen in the architectural layouts in Figure 50and Figure 51 registered codes are pointing at a Database. This will be handled by the TestCal system, as all sensor IDs will be passed to that system. However, this is not the only data being saved by the system. To monitor how the system behaves in the future, any tray that is not fully detected will be saved locally along with all of the sensor IDs in the form of a text log. The images will be named the date and time the image was taken, so that the information can be easily seen. The folders in which the images and ID logs are saved to can be changed by an Admin, this can be done monthly or quarterly depending on how often Movella wishes to change or review the logs.

6.1.2 Languages & Frameworks

The main programming language of this project is Python in both the front and back end. All subsequent libraries and frameworks are either already based in Python or have a python wrapper to allow them to be used by the system. For example, the Pylibdmtx library is a Data Matrix Library with a Python wrapper. The Framework used for the training and implementation of the Cascade Object detector is OpenCV. PyQt is the GUI framework used by the program to create an easy-to-use GUI.

6.1.3 System Work Flow

Now that the system has been properly defined, the internal Classes and GUI should be outlined. Knowing how the operator will move through the system will help in creating the class diagram of the system. Knowing that the operator will need to start by taking a picture of the tray of sensors before the Detection can begin, so this will be the starting point. Once the Object Detection algorithm returns its findings the operator will need to know whether all sensors have been detected. Should an error occur in the detection the operator will need to resolve the issue before beginning the detection again. The flow of the system can be seen in Figure 52.

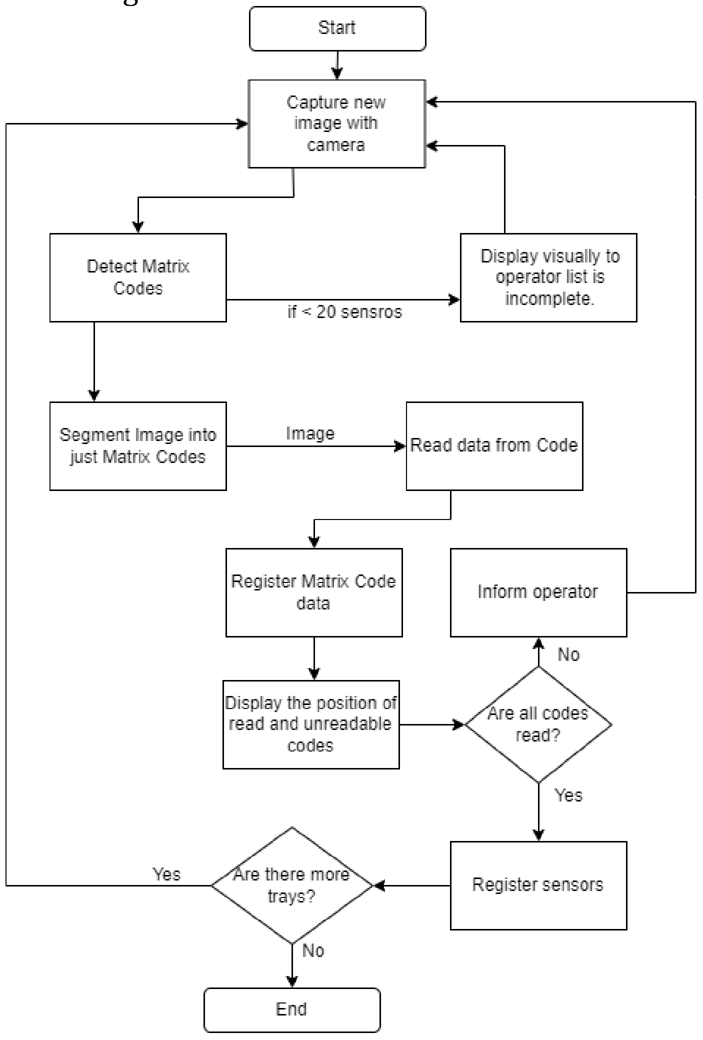
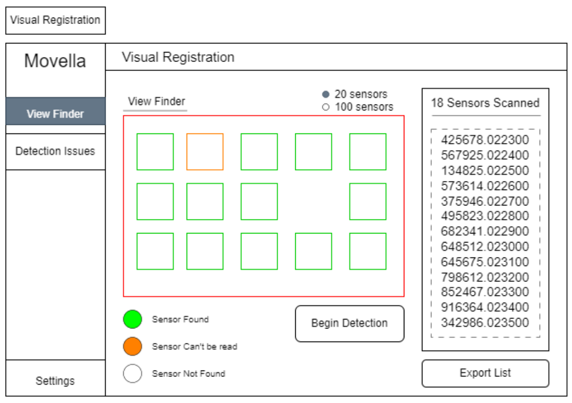

Start
Capture new image with camera
Detect Matrix Codes
if < 20 sensros
Display visually to operator list is incomplete.
Segment Image into just Matrix Codes
Image
Read data from Code
Register Matrix Code data
Inform operator
No
Display the position of read and unreadable codes
Are all codes read?
Yes
Yes
Are there more trays?
Register sensors
No
End

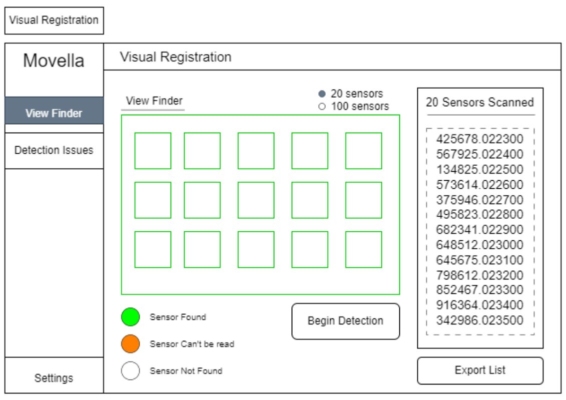
Figure 52 Flow Chart of User Interaction

From this flow chart, a wireframe of the GUI can be built. This is the first iteration of the design and will change as feedback is received from the production operators. First the Main page of the system will need to have a space where the sensor tray can be seen, as well as an area to see the output of the ID's once the sensors have been read. Lastly, a key to define what the different annotations on the image after the detection is shown. This will make it clear to the operator which sensors have been read, and which sensors need to be taken care of. The following images show the wireframe of the main page in a good and bad detection example.



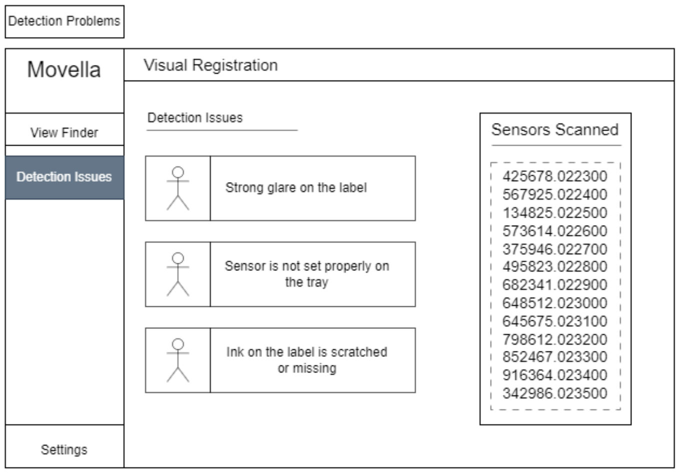
425678.022300
567925.022400
134825.022500
573614.022600
375946.022700
495823.022800
682341.022900
648512.023000
645675.023100
798612.023200
852467.023300
916364.023400
342986.023500

Figure 53 GUI diagram of bad sensor detection



425678.022300
567925.022400
134825.022500
573614.022600
375946.022700
495823.022800
682341.022900
648512.023000
645675.023100
798612.023200
852467.023300
916364.023400
342986.023500

Figure 54 GUI diagram of good sensor detection



425678.022300
567925.022400
134825.022500
573614.022600
375946.022700
495823.022800
682341.022900
648512.023000
645675.023100
798612.023200
852467.023300
916364.023400
342986.023500

Figure 55 GUI diagram of the Detection Issue Page

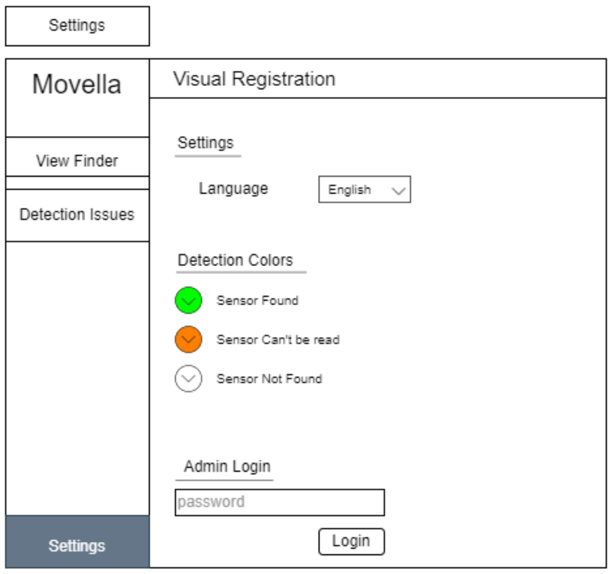


Figure 56 GUI diagram of the Settings Page

6.1.4 Code implementation

Now that the system has been designed, the implementation of that design can be addressed. The front and back end have two main functionalities, this being detecting where the DM code is, and extracting the data from the codes. Until the Detection algorithm is properly trained to work 100% of the time, we will need to be able to test if extracting the data from the code is working correctly. So, the Object Detector Class is an abstract class that has two implementations. One being the Cascade Detector (using the Object Detection Model), and the other being a pre-annotated class. The pre-annotated class is used to read the position of sensors in an image where they have already been identified, which acts as a work-around to test the system without using an Object Detection Model. In the Figure 57, you can see how the two classes inherit from the Object Detector Class. The other classes in the class diagram are the ReadMatrix class which implements the pylibdmtx library, and the MTiDetection class which stores all of the data associated with the MTi Sensors. This being the size and location of the detected sensors relative to the tray image, the data extracted from the sensor's DM code along with if it was able to be read by the pylibdmtx library, and finally the cropped image of the sensor. This image is saved with the sensor's data so that it can be recalled later. For example, if the sensor cannot be read, the sensor is displayed to show the reason it wasn't read i.e. the code was cut off or has a strong glare that renders the code unreadable.

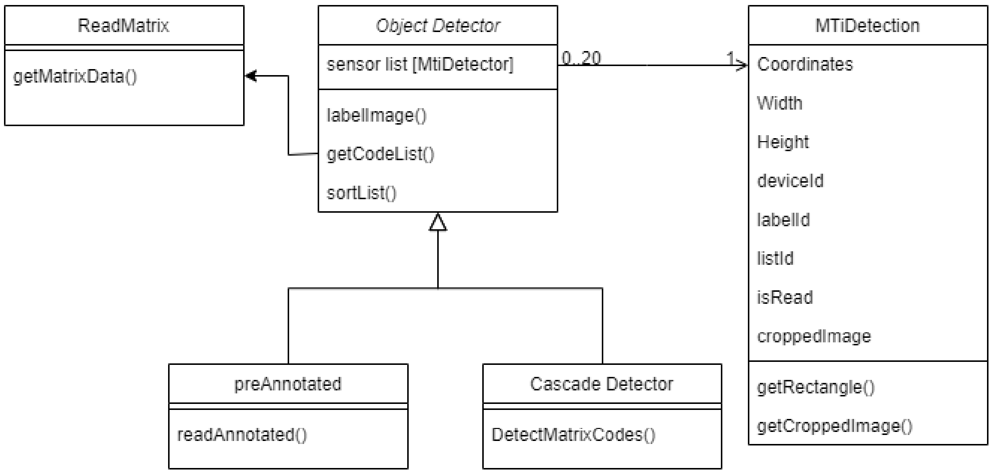


Figure 57 Class Diagram of the Visual Registration System

Within the code of these classes there are some Design patterns that were followed to ensure a standard was met. Iterators were used to walk through a list of sensors and used to help sort the detected sensors as they are not always detected in order from right to left.

The Factory Pattern was used to reduce the amount of code that was repeated throughout the system. This design pattern was used most notably in the Object detector class with the different approaches to the detecting sensors. This method was also used in the form of helper functions. These functions were used across the different classes from a single helper file. Helper functions were created if similar code was being used by different parts of the system. Displaying an image for example was used by multiple classes but all contained identical code.

The last design pattern used in the system is known as strategy, where an algorithm is wrapped inside of a class. This is done in both the ReadMatrix Class and the two child classes of the Object Detector. Each of these classes act as a wrapper for the algorithms they use.

6.1.5 Environmental Setup (lighting, camera position/angle)

Testing the camera structure was done in the intern office space, a well-lit room with natural sunlight coming in from one side of the room. The system was positioned on a desk with overhead lighting which provided enough light to clearly see the sensors but caused a strong glare on the DM code labels. To defuse the overhead lighting, a temporary plastic tray lid was used to avoid glare.

The production lab also had bright desk lights that can be turned on or off depending on what best suits the system. It is recommended to create some sort of light shielded to disperse strong lighting to avoid glare from the DM code labels.

Due to the intern office as well as the production lab having some amount of outside light influencing how bright the room is, training and test data were captured on days with varying weather conditions, to ensure the system can work any day of the year no matter the external weather conditions.

The camera was positioned directly above the tray on the Camera mount structure.

6.2 Integrating CV System into GUI

With the system design outlined and an Object Detection Model created, the final set is to apply the Graphical User Interface to the ODM model. Following the Wireframes outlined in 6.1.3, the GUI was created using PyQt.

The Front end of the system contains a view port which shows the images captured by the Raspberry Pi. In the current implementation, the RPi is controlled from the computer Through a program called VNC viewer, which provides remote access. The captured image can be transferred to the PC through a program called SSHFS (Secure Shell File Transfer) which connects a mounted file to the PC to share images. While this is not the most efficient method to transfer the data, it does get the system running. For the final implementation of the system the RPi will be set up as a stream where the GUI displays the camera in real time.

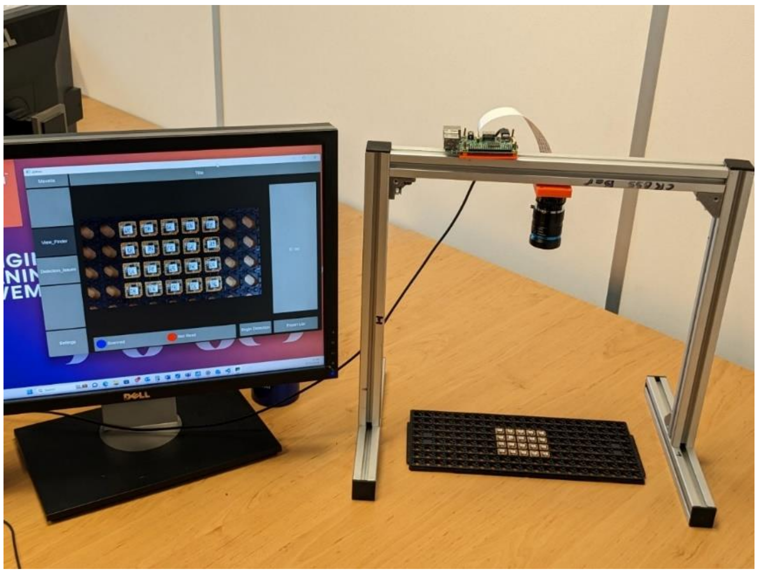


Figure 58 Full System with GUI

Upon receiving the captured image, the back end of the system can analyse the image. This is done by handing the image to the ODM model using the OpenCV Library which then returns the list of detected sensors and their positions. The back end saves each sensor as an MTiDetection(Figure 57 Class Diagram) and creates a smaller cropped image from the original captured image. Every detected sensor is then scanned by the pylibdmtx library and returned to the list (Appendix of the back end code).

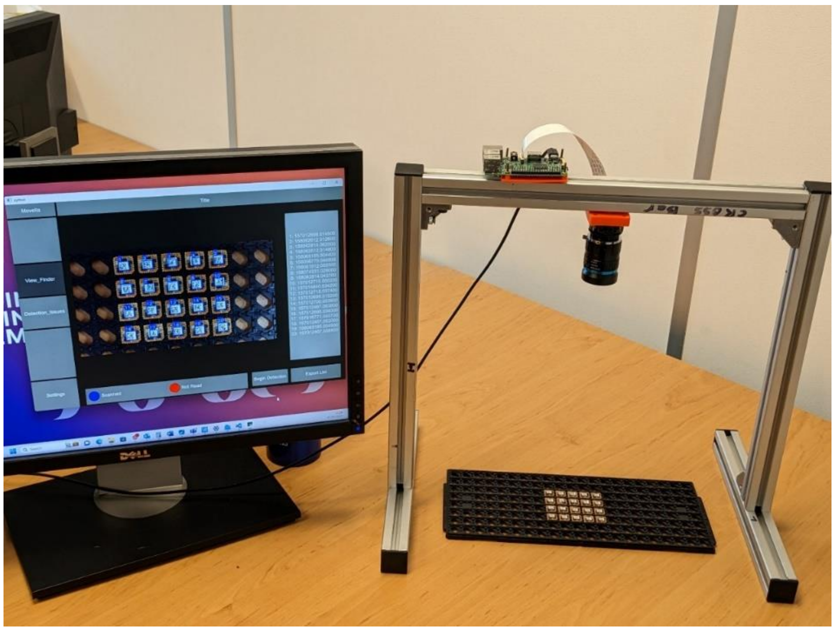
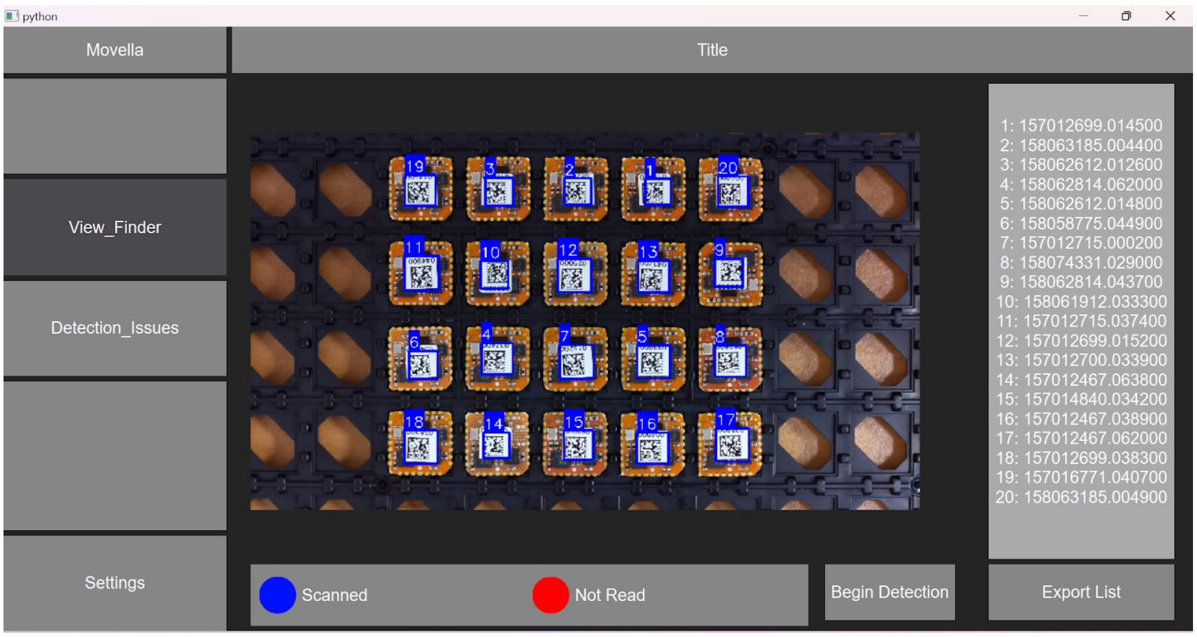


Figure 59 Full System with Tray of 20 sensors, Mounted camera, GUI and ODM output

During the integration of the GUI to the ODM model, was ensuring the output list matched the annotation from the model. Operators wanted a clear indication of which ID in the list corresponded to the detected sensor in the image, and requested that it be read from left to right.

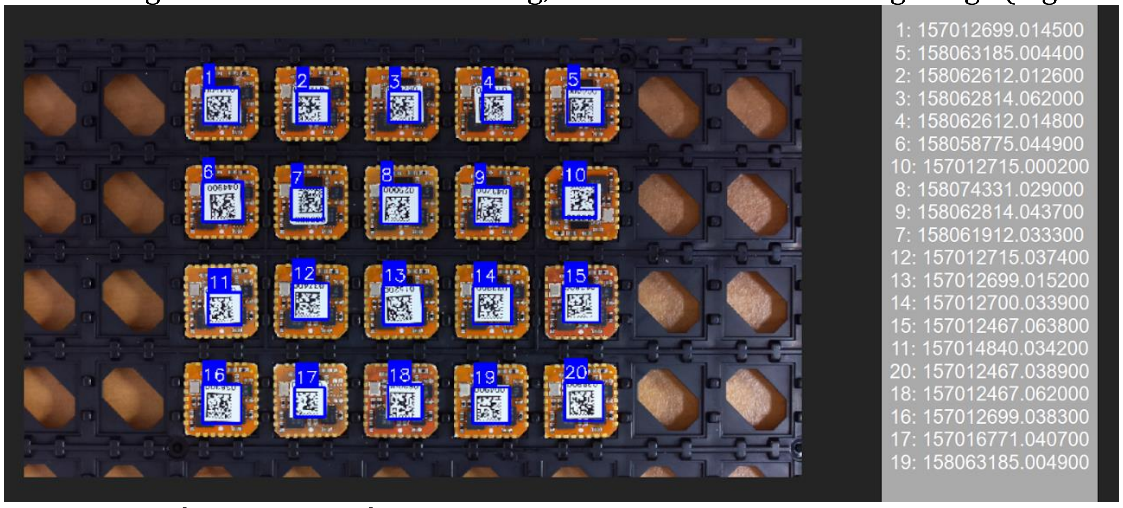
However, the Object Detector does not return the list of sensors from the top left to the bottom right, but instead returns the list in the order the Sensors were identified (Figure 60). This made it very difficult to figure out which ID corresponded to which Sensor, as the number on the list is the order of which the codes are in the list, and do not match the annotated codes. (To improve readability, the navigation bar will be cropped out from subsequent figures)



1: 157012699.014500
2: 158063185.004400
3: 158062612.012600
4: 158062814.062000
5: 158062612.014800
6: 158058775.044900
7: 157012715.000200
8: 158074331.029000
9: 158062814.043700
10: 158061912.033300
11: 157012715.037400
12: 157012699.015200
13: 157012700.033900
14: 157012467.063800
15: 157014840.034200
16: 157012467.038900
17: 157012467.062000
18: 157012699.038300
19: 157016771.040700
20: 158063185.004900

Figure 60 GUI without Annotation Matching List IDs

Using the listId variable from the MTiDetection class it is possible to connect the id of the annotated code to the ID of the sensor in the list. However, this causes the list to appear to be unorganised and can be confusing; as shown in the following image (Figure 61).

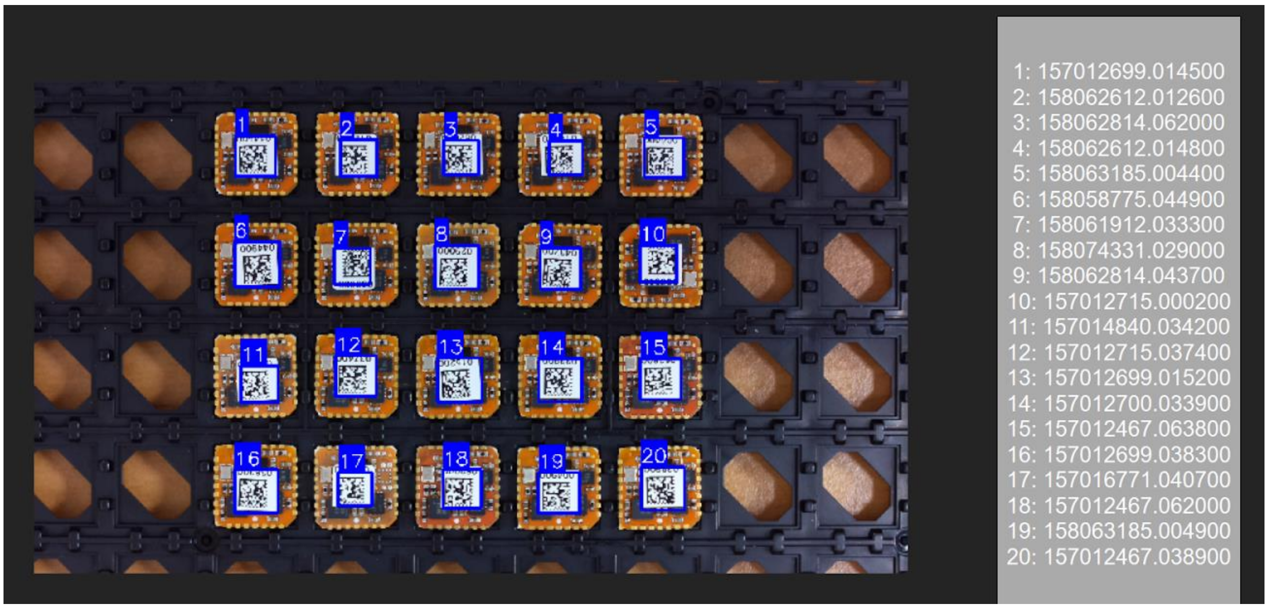


1: 157012699.014500
5: 158063185.004400
2: 158062612.012600
3: 158062814.062000
4: 158062612.014800
6: 158058775.044900
10: 157012715.000200
8: 158074331.029000
9: 158062814.043700
7: 158061912.033300
12: 157012715.037400
13: 157012699.015200
14: 157012700.033900
15: 157012467.063800
11: 157014840.034200
20: 157012467.038900
18: 157012467.062000
16: 157012699.038300
17: 157016771.040700
19: 158063185.004900

Figure 61 GUI with Annotation Matching List IDs

To solve this the list is sorted based on the position of the identified sensor. This could then be sorted from top to bottom and left to right. However, the rows should contain 5 sensors when doing a tray of 20 sensors, but it is possible that a double detection, or a false positive occurs. In which case, it was possible that the last sensor in the row is sorted into the next row, which caused a discrepancy between the list and the sensors.

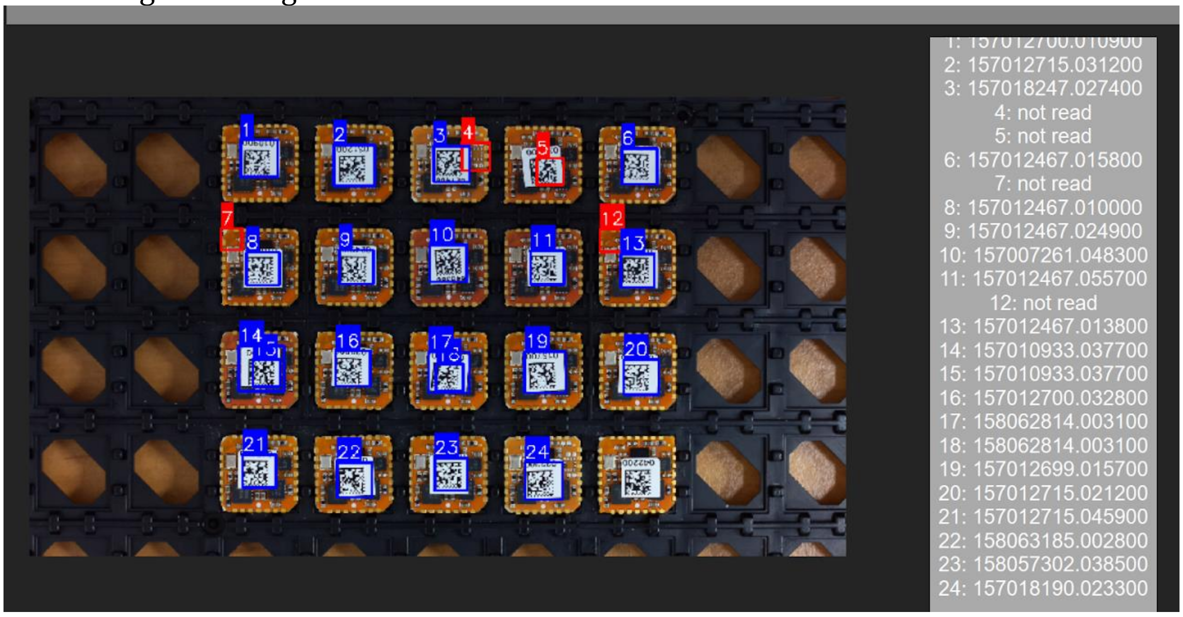
To fix this, rows were categorised into clusters based on their y coordinates, with a maximum threshold, should the coordinate be over this threshold, it is then known for certain that the sensor is in a different row, and a new cluster should be created. This adaptive row clustering allowed for the list to be fully sorted and correctly labelled so that it was clear to the operator which sensor was which. (Code for the sorting algorithm can be found in Appendix C.1 Object Detector)



```
1: 157012699.014500
2: 158062612.012600
3: 158062814.062000
4: 158062612.014800
5: 158063185.004400
6: 158058775.044900
7: 158061912.033300
8: 158074331.029000
9: 158062814.043700
10: 157012715.000200
11: 157014840.034200
12: 157012715.037400
13: 157012699.015200
14: 157012700.033900
15: 157012467.063800
16: 157012699.038300
17: 157016771.040700
18: 157012467.062000
19: 158063185.004900
20: 157012467.038900
```

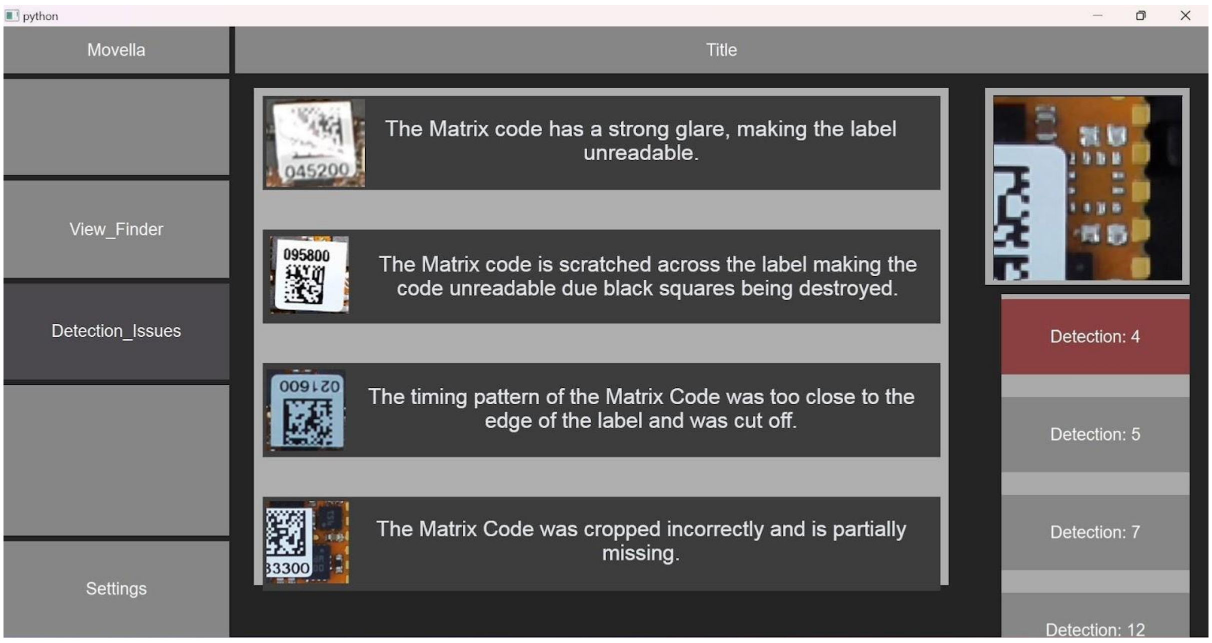
Figure 62 GUI with Sorted Annotation and List IDs

The last part to focus on with regards to integrating the Detection model into the GUI is ensuring the system still worked in not only good weather tests, but also bad weather examples. The following images are screenshots of GUI when something with the detection goes wrong.



```
1: 157012700.010900
2: 157012715.031200
3: 157018247.027400
4: not read
5: not read
6: 157012467.015800
7: not read
8: 157012467.010000
9: 157012467.024900
10: 157007261.048300
11: 157012467.055700
12: not read
13: 157012467.013800
14: 157010933.037700
15: 157010933.037700
16: 157012700.032800
17: 158062814.003100
18: 158062814.003100
19: 157012699.015700
20: 157012715.021200
21: 157012715.045900
22: 158063185.002800
23: 158057302.038500
24: 157018190.023300
```

Figure 63 GUI Handling Detection Errors



The Matrix code has a strong glare, making the label unreadable.

The Matrix code is scratched across the label making the code unreadable due black squares being destroyed.

The timing pattern of the Matrix Code was too close to the edge of the label and was cut off.

The Matrix Code was cropped incorrectly and is partially missing.

Figure 64 GUI Showing High Res Image of a Detection Issue (Detection 4)

6.3 CV System Demonstration

During the product development process, a number of system demonstrations were given. Most of these demonstrations took place during the research and development phase of the project and typically focused on the performance of two or more integrated system components, such as the integration of a camera and a camera mounting platform to take high quality sensor pictures, or the integration of the ODM and the pylibdmtx library to obtain readable DM code images.

Other demonstrations occurred during and at the end of the prototype development phase. Like the R&D demonstrations, the Prototype demonstrations typically focused on system capabilities. So, where the R&D demonstrations presented the capabilities of the integrated camera system, and the integrated ODM system, the Prototype demonstration highlighted the integration of the ODM output with the GUI.

Finally, the operation of the fully integrated CV system was demonstrated at the end of the project when all system components had been identified and integrated into the final product. The demonstration presented the product's features and capabilities, and the system's value to the end user. Like the R&D and Prototype demonstrations, the fully integrated system demonstration was made to project's stakeholders, specifically the Movella Production and R&D managers during one of the weekly meetings.

6.4 CV System Testing

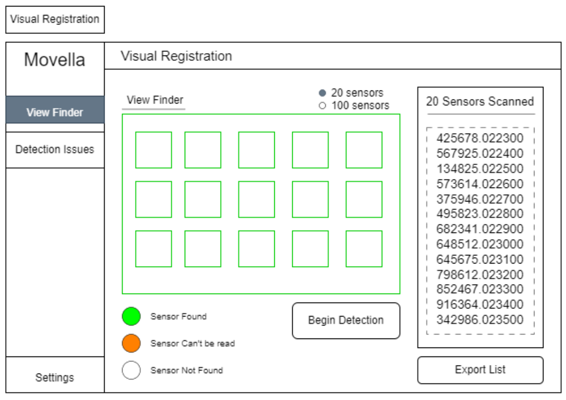
Having tested the CV system's individual components it was time to test the integrated CV system to see if the system's performance met the system's customer requirements. A number of approaches are available to determine if the product functions as expected (QA Testing) and if customers / users find value in using the product (Concept Testing).

Quality assurance (QA) testing is typically undertaken in a staged environment, where tests are run to determine the functionality of the product before moving on to another state of development till the product is completed. This type of product testing ensures that the product works as expected and helps researchers identify problems before making the product available to the user, customer or general product.

In this project, QA testing was used (similar to integrated system demonstration) throughout the development of the CV product. Typically performance tests were run prior to one of the weekly stakeholder meetings. The results of the test were then discussed and a decision was made to either move ahead to the next stage of system development or to solve the problem(s) causing issues.

Concept testing occurred at the end of the product development phase and is typically performed to provide clarity on certain features of the product. or functionality customers want from the product. Concept testing often involves presentations, surveys / interviews, and wireframes to obtain feedback from users / customers.

In this project, both wireframes and surveys were performed to obtain feedback from machine operators on the use of GUI to communicate system errors and system functionality (Figure 65). Both were used to provide clarity on the use and functionality of the CV system as a tool to automatically scan and register MTi sensor DM codes.



```
425678.022300
567925.022400
134825.022500
573614.022600
375946.022700
495823.022800
682341.022900
648512.023000
645675.023100
798612.023200
852467.023300
916364.023400
342986.023500
```

Figure 65 Example of a Wireframe diagram used

6.5 CV System Evaluation

While it was envisioned that the functionality of the conceptual product would be tested in the Movella production facility, this did not happen. One of the offices was chosen instead. While it is believed that the location of the testing site did not impact the overall functionality of the CV system, it may have impacted the outcome of some of the tests.

The reason for this being that the lighting conditions in the lab (fluorescent light) were different from the lighting conditions in the office (incandescent light). As DM code images are covered by a shiny coat, glare was often a problem in the scanning of sensors, particularly in the production fab (Fig. xx). While this problem was easily fixed in the office, the installation of a light diffuser shield in the production lab might have proved more difficult.

As such, it is possible that if sensor detection tests had been conducted in the production lab rather than the office, the robustness of the ODM might have improved as the number of unreadable images due to glare would undoubtedly have increased the number of training samples and subsequently reduced the image detection time. A downside to this increase in system robustness would also be an increase in sensor code registration time as for all faulty sensor images detected, the whole tray of sensors would have to be rescanned, even if there only was one faulty sensor on the tray.

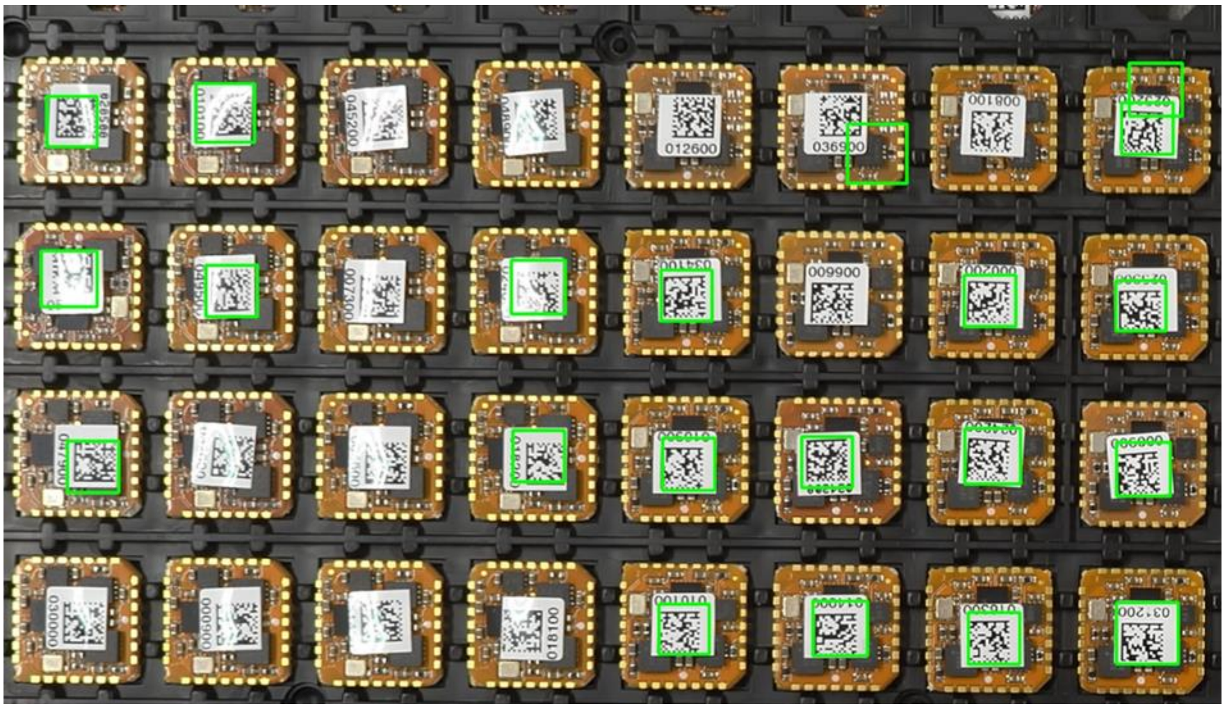


Figure 66 Tray with sensors showing object detection and image glare.

As such, it is possible that if sensor detection tests had been conducted in the production lab rather than the office, the robustness of the ODM might have improved as the number

of unreadable images due to glare would undoubtedly have increased the number of training samples and subsequently reduced the image detection time. A downside to this increase in system robustness would also be an increase in sensor code registration time as for all faulty sensor images detected, the whole tray of sensors would have to be rescanned, even if there only was one faulty sensor on the tray.

7 Conclusion

Data Matrix (DM) codes offer high density data in small sizes. They are ideal candidates for use on products that have a limited surface to print on. Yet, their small size and limited number of DM decoding devices. During the internship, development of a tool that can detect and process DM codes in high numbers. Specifically, the development of a conceptual computer vision system that can automatically register DM codes. Data show that it is feasible to implement the tool in manufacturing.

Requirements Met:

The program can detect the positions of multiple Matrix Codes within an image.

- Tested by running the system and comparing the IoU intersection.

The program can read the data of a Matrix Code with 100% correctness.

- Tested by comparing the output of the system with a reliable DM code reader.

The system can process a tray of sensors without registering the same code twice.

- Double detections are filtered out by the system before exporting the ID list.

There is a GUI an operator can use for the process, to see if action is required.

- The Detection Issue page displays a high-res image of a detected sensor that cannot be read.

The system can reliably register the sensors and is robust enough to handle missing Matrix Codes, crooked sensors (sensors not properly set on trays), and other outlying situations.

- The Object Detection Model can detect the DM codes on sensors shown to it, and it can handle outlying situations it's never seen before. It still has some short coming in detecting sensors with glare and still has a chance of creating a false positive. However, The system does makes up for this by handling the error or presenting it to the operator.
- To fix the false positives and missed sensors in certain lighting conditions, targeted training data can be added to the model with some extra training stages the model should be able to rectify this short coming.

The system takes the same amount of time or less than a trained operator scanning a tray (30 seconds for a tray of 20).

- The system is capable of reading a tray of 20 sensors in 5 seconds on average. Which is 6 times faster.

Bibliography

Ben. (2023). *Training a Cascade Classifier*. Retrieved from Learn code by gaming: https://learncodebygaming.com/blog/training-a-cascade-classifier

Boesch, G. (2023). *Object Detection in 2023*. Retrieved from Viso.ai: https://viso.ai/deep-learning/object-detection/

Data Matrix. (2023). Retrieved from Apose: https://docs.aspose.com/barcode/info-cards/data-matrix/

Habibi, H. A. (2017). *Guid to convolutional neual networks.* Cham, Switzerland: Elnaz Jahani Heravi.

Haije, E. G. (2023). *Customer Feedback tools*. Retrieved from mopinion: https://mopinion.com/customer-feedback-tools/

Kanban Methodology. (2023). Retrieved from Wrike: https://www.wrike.com/kanban-guide/what-is-kanban/

Karrach, L. (2021, August 27). *Comparative Study of Data Matrix Codes Localization and Recognition Methods*. Retrieved from MDPI: https://www.mdpi.com/2313-433X/7/9/163

Laghton, M. (2019). *libdmtx*. Retrieved from sourceforge: https://libdmtx.sourceforge.net/

McCue, I. (2022, Septamber 5). *Barcodes Defined*. Retrieved from NetSuite: https://www.netsuite.com/portal/resource/articles/inventory-management/barcode.shtml

Mesquita, D. (2021, May 28). *Object Detection Evaluation*. Retrieved from towards data science: https://towardsdatascience.com/introduction-to-object-detection-model-evaluation-3a789220a9bf

Movella About Page. (2023). Retrieved from Movella.com: https://www.movella.com/company/about-us

Mti Product Selector. (2023). Retrieved from Movella.com: https://www.movella.com/products/sensor-modules/xsens-mti-product-selector

MTi-1 IMU. (2023). Retrieved from Movella.com: https://www.movella.com/products/sensor-modules/xsens-mti-1-imu#overview

Nanos, G. (2023, January 24). *Machine Learning Flexible and Inflexible models*. Retrieved from Baeldung: https://www.baeldung.com/cs/ml-flexible-and-inflexible-models

Norton, N. (2023). *How to Reduce Camera Shake*. Retrieved from Digital Photography School: https://digital-photography-school.com/how-to-avoid-camera-shake/

Object Detection Model Evaluation. (2021, May). Retrieved from towardsdatascience.com: https://towardsdatascience.com/introduction-to-object-detection-model-evaluation-3a789220a9bf

Oney, L. (2018, Jan). *GS1 Data Matrix Guideline.* Retrieved from GS1: https://www.gs1.org/docs/barcodes/GS1_DataMatrix_Guideline.pdf

OpenCV Cascade Clasifier. (2023). Retrieved from OpenCV.com: https://docs.opencv.org/4.x/db/d28/tutorial_cascade_classifier.html

OpenCV Cascade Classifier Training. (2023). Retrieved from OpenCV: https://docs.opencv.org/4.x/dc/d88/tutorial_traincascade.html

OpenCV QR Code Detector. (2023). Retrieved from OpenCV: https://docs.opencv.org/4.x/de/dc3/classcv_1_1QRCodeDetector.html

OpenCV Template Matching. (2023). Retrieved from OpenCV: https://docs.opencv.org/3.4/d4/dc6/tutorial_py_template_matching.html

Pedamkar, P. (2023). *What is GUI*. Retrieved from EDUCBA: https://www.educba.com/what-is-gui/

Pylibdmtx library. (2022). Retrieved from GitLab: https://github.com/NaturalHistoryMuseum/pylibdmtx/

Regression Testing. (2023). Retrieved from Katlon: https://katalon.com/resources-center/blog/regression-testing

Sensor Modules. (2023). Retrieved from Movella.com: https://www.movella.com/products/sensor-modules

Stazzone, S. (2023, January 17). *How Barcodes Work to Increase Efficiency*. Retrieved from Metal Photo of Cincinnate: https://www.mpofcinci.com/blog/how-barcodes-increase-efficiency/

Thingiverse. (2023). Retrieved from ThingiVerse: https://www.thingiverse.com/thing:4595863/files

training a cascade classifier. (2020, August 22). Retrieved from learncodebygaming.com: https://learncodebygaming.com/blog/training-a-cascade-classifier

Vandendorpe, A. (2021, April 28). *Choosing a Camrea for Machine Learning*. Retrieved from ml6.eu: https://blog.ml6.eu/how-to-choose-a-camera-for-ml-e2a1819f37e0

Appendix

A.1 Two Dimensional Barcode Reader

Figure 67 Laser Gun

A.2 Reading Data Matrix Codes

Inside the data section (the black and white cells surrounded by the finding and timing patterns) are smaller sub sections of L like squares that take a 3 x 3 space but are missing the top right cell. In Figure 6868 and Figure 6969 an example of how these data cells look is shown, These cells demonstrate how a byte of data in binary is related to the layout of the DM code's data cells. Once translated to binary they are then converted to Ascii (More info on Ascii tables in Appendix X).

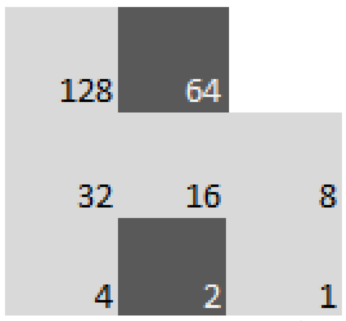


Figure 68 A Matrix data section representing 'A' on the Ascii Table.

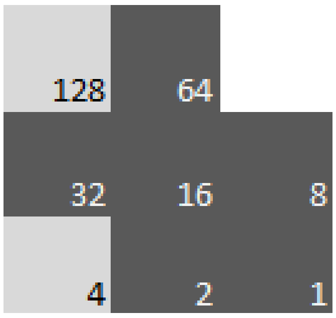


Figure 69 A Matrix data section representing 'z' on the Ascii Table

The way these data sections are shaped allow for a compact way to fit them into the data matrix code, where the bottom left cell is positioned in the empty space of another data section. Figure 70 shows how these data sections are packed into a DM code. The green sections are where the data of the code is stored, the red is the error correction (which makes the code reliable), yellow is the buffer between those sections, and orange tells the decoding process when to stop reading.

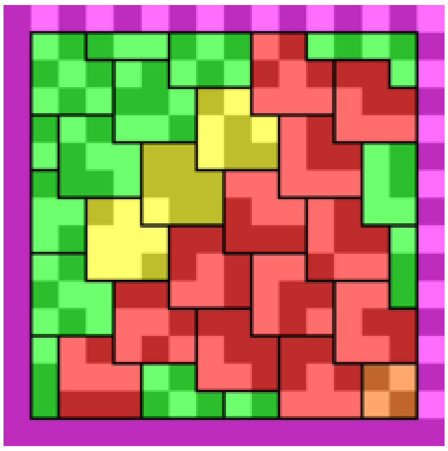


Figure 70 Data Matrix with colour labelled data sections Sourced From Wikipedia.

Figure 71 illustrates the encoding process of a DM code. While encoding, the Least Significant Bit (the cell labelled “1” in Figure 68 and Figure 69) acts as the anchor point of each section. As the matrix is encoded it follows a diagonal zig zag pattern through the space, placing each data section at its anchor point.

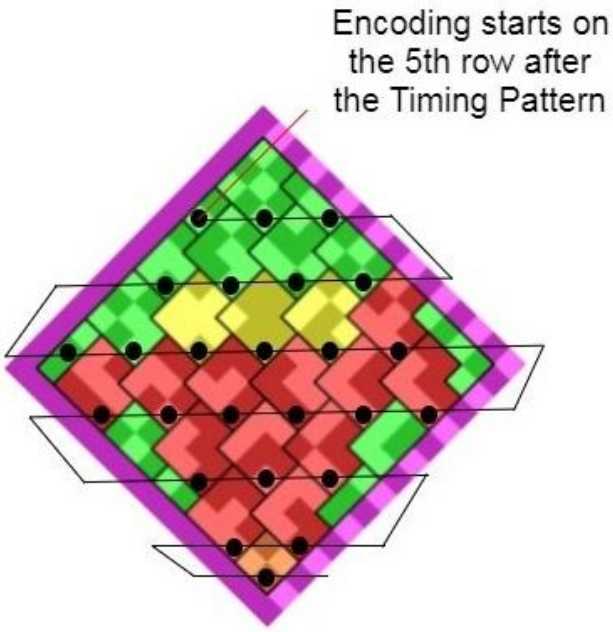


Figure 71 Annotation over original image of a Data Matrix encoded process.

Figure 70 and Figure 71 show that some sections near the timing pattern do not match the standard shape of the data section, this is due to the sections intersecting with the border pattern. In Figure 71 the first data section intersects with the timing pattern; this is known as an edge intersection. This occurs when the section intersects with either the side or top of the data area, the data section is then wrapped to the other side of the data area (Shown in Figure 72).

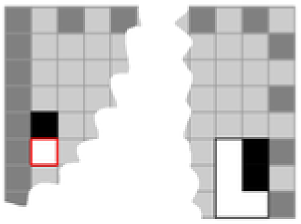


Figure 72 Example of a left side intersection Sourced from Wikipedia

When a data section is wrapped to the other side of the data area, the section is translated either right or down 2 squares depending on if it intersects with the side, or with the top. The exception of sections only shifting 2 squares and maintaining their shape; is when it intersects with a corner. When this happens, maintaining the square shape is no longer possible and must be specially manipulated to fit within the available space. There are four possible cases depending on the size of the DM code and how the sections align (Oney, 2018). Due to this being a special case, the corner data placement is resolved after the rest of the DM code has been encoded. This ensures that all sections can be encoded correctly. Since the matrix is filled in diagonally, it is only possible to intersect with the corner once per code.

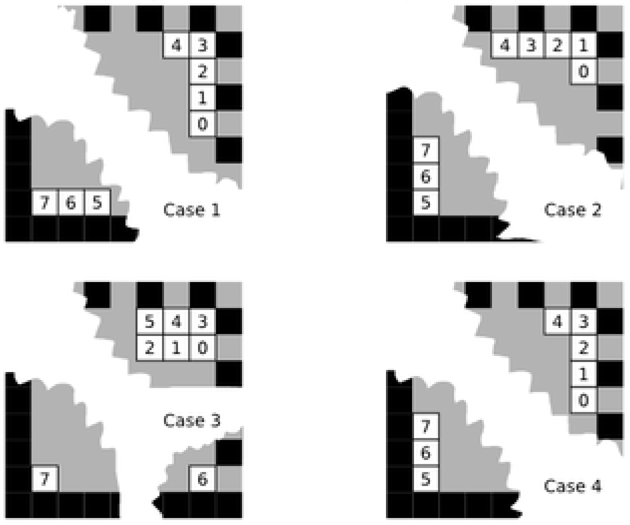


Figure 73 Corner intersection case solutions sourced from Wikipedia

Now that a basic understanding of encoding DM codes has been stated, reading the codes can be done by deciphering the data squares in the same manner. While this could be programmed to be done an algorithm, it would take a lot of time. So, a library or tool that is capable of reading DM codes will be selected to help start the project. The library selection will be explored further during the research phase in section Question: Which tools are needed, and what physical setup is required, to create a CV system that can successfully read DM barcodes?4.2.

B. Confluence

B.1 Work Log

Week Nr.	Date	Notes	Questions / Points to be discussed
1	25 Nov 2022	Added a Visual Registration page to my Student space, and updated it with the research of Matrix Codes from this week, as well as test conducted with the Logitech C270 WebCam.	The Proof of Concept is limited to a small data pool, it confirms that it is possible to read Matrix Codes with the camera but does not cover all potential cases. (ie. distance, lighting, rotation of code, etc.)

2	28 Nov 2022	Updated Visual Registration page with examples and updated research for Algorithmic challenges. Worked with the Realsense WebCam on another proof of concept.	
2	29 Nov 2022	Updated Visual Registration page with the proof of Concept from yesterday and added project roadmap. Worked with the RealSense Camera and increased the efficiency of reading the Matrix Code from an 8 second run time down to 0.3 seconds.	Noticed an interesting behavior with the Adaptive Gaussian Thresholding outlined in the Visual Registration page. When increasing the clarity of the Matrix code through image manipulation, the pylibdmtx library had a harder time reading it than a murkier result.
2	30 Nov 2022	Visited Production and was introduced to how the current sensor registration process is handled and what measures employees take to ensure the quality of the sensors (ESD protection). Presented the first week's progress to Fabian and Dieuwe during weekly meeting	
2	1 Dec 2022	Visted Production again to finish watching the registration process of sensors. Continued research and testing in preparation for the matrix code Dieuwe will give me to test.	
2	2 Dec 2022	Received old MTi 1 sensors from Dieuwe for testing and worked on getting a clear	

		readable image from the Matrix Codes on each sensor.	
2	3 Dec 2022	Continued work on reading the Matrix Codes and determined the RealSense Web Cam couldn't capture a clear enough image to read the Matrix Code and suggested the use of a Macro Camera to counter the small size of the Matrix Codes.	
3	5 Dec 2022	Experimented with the Xsens macro camera & lens as well as the C920 Pro lens and was able to get a readable Matrix Code Image of the sensor from both cameras. However, both were unable to get readable results of more than 6 sensors at a time which is not enough for the goal of the project. A new camera with a high resolution and camera lens is currently being ordered to better match the requirements.	
3	6 Dec 2022	Researched training object detection models and what steps are needed to make each method work.	
3	7 Dec 2022	Updated wiki with the last weeks camera testing. Met with Fabian about the last weeks progress and what steps to take before our Meeting with Rogier (School Supervisor) on the 14th of December. Researched Neural Networks.	
3	8 Dec 2022	Continued research into Neural Networks and how they	

		compare to traditional machine learning methods.	
3	9 Dec 2022	Started collecting useable Matrix Code images of the MTi-1 sensors for the dataset to train the cascade classifier. Worked on Plan of Approach Documentation.	
4	12 Dec 2022	Submitted Plan of Approach rough draft to Fabian and applied recommended changes. Worked on presentation for Wednesday.	
4	13 Dec 2022	Added finishing touches to PoA and submitted it to Rogier. Finalized the presentation for Wednesday.	
4	14 Dec 2022	Met with Dieuwe, Fabian and Rogier and discussed the Plan of Approach and work done so far.	From the meeting, it was determined to add Research Questions to the PoA, and for the future documentation be sure to explain why/motivation for using a framework or method.
4	15 Dec 2022	Finished collecting preliminary training data for the cascade model for me to work on while working from home.	
5	19 - 23 Dec 2022	Worked on training the cascade model and looked into what parameters could be adjusted to get the best result.	
6	26 - 30 Dec 2022	Continued work on training the cascade model, and worked on the project requirements and research questions.	

7	5 Jan 2023	Updated the wiki space on the work done so far and wrote a break down the cascade model covering what works and why some issues were occurring.	
7	6 Jan 2023	Added more negative examples to the training data to target the false positives that were occurring in the model before. Worked on another approach with the training data to identify the sensors rather than the Matrix Code labels to see if it increased it's accuracy.	
8	9 Jan 2023	Updated the Cascade Object Detection documentation to include the results of the the sensor model approach. Received Raspberry Pi & camera from Fabian, and started work on the mechanical setup to support the camera above the sensor trays.	
8	10 Jan 2023	Cut two aluminum T-slot rods into an array of rods and assembled them with corner connectors to create the physical set up for the camera. Started working with the Raspberry Pi and looked into how to attach the camera module to the physical structure.	
8	11 Jan 2023	Weekly meeting with Fabian and Dieuwe, which covered the progress of the Visual Registration Cascade Model so far and what approaches could help improve it. We also	

		covered the research questions and decided to broaden them for the Plan of Approach. Started working on an approach to reduce the number of false positives coming from the cascade model.	
8	12 Jan 2023	Started working with the Raspberry Pi to begin testing the program so far. Wrote a helper program to expand the number of training data samples by rotating each image by 90 degrees 3 times to help with false negatives.	
8	13 Jan 2023	Met with Rogier and caught him up with the progress made since their last visit and planned on what documentation should be worked on next and planned our next meeting for the 27th of January. Continued work on importing the current code to the Raspberry Pi and started getting the Pi ready to work with an on board camera.	
9	16 Jan 2023	The object recognition program takes a little while to run on the Raspberry Pi, so I started training a model with a 48x48 window instead of a 24x24 to see if this would help decrease the runtime on the Raspberry Pi. While this approach takes longer to train, it may make the algorithm more efficient and cover larger images in a shorter amount of time.	

9	17 Jan 2023	Documented the results of the the 48x48 window training (not much difference on the PC, but improved the models processing speed on the Raspberry Pi). Updated the file structure of the Visual Registration program to make it easier to transfer between computers and easier to update in the future. Worked on the internship documentation in preparation for the mid-term of the project.	
9	18 Jan 2023	Had a weekly meeting with Fabian where we went over the research questions and what needs to be done for the mid-term documentation. As well as what the next week of the project will look like Attached the camera to the structure and started collecting some images of the filled sensor trays, and seeing how the trained model reacts to a tray of sensors.	
9	19 Jan 2023	Worked on the Documentation for the mid-Term from home.	
9	20 Jan 2023	Worked on the Documentation for the mid-Term and started working on the 3D concept for the camera mount model from home	
10	23 Jan 2023	Gathered measurements and started working on the 3D camera mount and worked on the annotation augmentation rotation program.	

10	24 Jan 2023	Connected to the Raspberry Pi over ssh and got the VNC connection working, and started working on mounting a folder to share data between devices. Started looking into how the current detection model responds to a tray of sensors in different lighting conditions for trays of 20 and trays of 100 sensors.	The camera has a wide angle lens, so when looking at a tray with 100 sensors, the furthest sensors start to look a bit distorted but are still recognized by the detection model.
10	25 Jan 2023	Met with Fabian and Dieuwe and talked about the progress of the last week and what needs to be focused on for next week. Started looking into the Matrix library and double checking it's validity, by comparing what the program read to the results of an external app on my phone.	Some interesting Matrix Codes were found, two sensors had the ID of: 00000000 and two other sensors had the same ID of: 157012699 with two different label ID's
10	26 Jan 2023	Printed the camera mount and prepared it to be attached to the physical structure, and updated the student space Use of the 3D printer wiki page. Finished getting the connection between the Windows computer and the Raspberry Pi working and it is now possible to share the files between them and work on the Pi remotely. Fixed bugs in the detection program.	
10	27 Jan 2023	Met with Rogier and talked about the mid-term Documentation and Requirements. Showed Demo of the project so far,	

		Worked on the Physical Structure attaching the Pi camera and Raspberry Pi. Continued work on the Documentation and tested the library's accuracy when reading Matrix Codes	
11	30 Jan 2023	Worked with the physical structure to try to get the images to be as clear as possible for testing the Matrix Code library. Began testing and validating the libraries accuracy. Worked on mid-term Documentation.	
11	31 Jan 2023	Talked with Fabian about the mid-term Documentation and what changes could be made to make it easier to understand for the reader. Worked on getting the best possible image from the physical set up. Looked into the accuracy of the library and if it was able to return the position of Matrix Codes.	
11	01 Feb 2023	Had a meeting with Fabian and Dieuwe where we discussed the progress of the last week, what Movella does, for the Documentation, and about moving into the planning phase of the GUI. Worked on the mid-Term documentation for submission.	

11	02 Feb 2023	Worked on getting the Library to read Matrix Codes at 100% accuracy in perfect conditions, and looked into the speed of the program when cropping the Images. Started looking into how to host a GUI web browser on the Raspberry Pi.	When running the program using only the dmtx library it takes 30 seconds to process a tray of 20 sensors. The pre-annotated sensors with cropping down to the matrix code took about 5 seconds to process a tray of 20 sensors. Which decreased the amount of time the program took by 80%
11	03 Feb 2023	Worked on proof of concept of a web server hosted on a Raspberry Pi and what tools could be used to accomplish this	
12	06 Feb 2023	Worked on data flow of the program and what possible interactions a user would need in order to go through the entire process. and graphed the results of the current program	
12	07 Feb 2023	Continued research of potential tools usable for the GUI and added a comparison graph of the training results of the Object Detection models	
12	08 Feb 2023	Met with Fabian and talked about the GUI so far and what needed to be planned next, we also went over the program so far and discussed what programming tools should be used to keep the code to standard.	
12	09 Feb 2023	Updated Architecture and worked on GUI wire frames. Updated the main program to follow coding standards.	

12	10 Feb 2023	Met with Rogier and talked about the progress of the last two weeks, and talked about the Documentation so far and what changes should be made. Met with Fabian and Dieuwe where we talked about the progress of the GUI planning. Which covered the GUI page ideas, data flow, UI Format and how the architecture could be improved.	
13	13 Feb 2023	Updated GUI Pages and the System Architecture, got feed back from Dieuwe and uploaded most recent versions to my student space under GUI Diagrams. Added AutoDoc documentation to the program and updated the github. Worked on researching tools that would be needed to implement the architectural approaches.	
13	14 Feb 2023	Researched tools to use for implementing the GUI and architectural approach. Worked on presentation for Wednesday's weekly meeting.	
13	15 Feb 2023	Met with Fabian and Dieuwe where I presented the work so far on the GUI and architectural approaches. Prepared to meet with operators to discuss the project and gather feedback on the GUI. Started working on the GUI.	

	16 Feb 2023	Prepared small presentation for the Production Operators and made a few changes to GUI. Worked on the GUI with PyQt and looked into how PyQt can be used to meet the needs of the GUI design	
	17 Feb 2023	Met with Dieuwe, Marco, and the Production Operators to gather feed back on the GUI as well as gathered information on what features they would like to see to make the Application easy for them to understand.	
	20 Feb 2023	Worked on GUI using PyQt and planned out what tasks should be added to the Scrum board for the first Sprint Updated Documentation after feedback.	
	21 Feb 2023	Filled Scrum back log with tasks and attached their weights and epics. Continued to implement the base GUI layout.	

B.2 Weekly Meeting Notes

Date	Notes	Goals for the next week
		• Wim Luyendijk make a proposal for planning of internship 30 Nov 2022 • Wim Luyendijk add picture to confluence 30 Nov 2022 • Wim Luyendijk visit production environment to see usecases 30 Nov 2022 • Wim Luyendijk understand and present basic functionality of Matrix code 30 Nov 2022

		• Wim Luyendijk look for relevant related work (software, papers, report) 30 Nov 2022 • Wim Luyendijk add short summarizing overview of related work to Wiki 30 Nov 2022 • Wim Luyendijk complete information on your student profile Wim Luyendijk 30 Nov 2022
7 Dec 2022	• Discussed Cameras tested so far. • Talked about Plan of Approach for next steps of the project. • Discussed Cascade model training and AdaBoost. ◦ Sectioned Research into Traditional Machine Learning ◦ Extended Research to incorporate Neural Networks to compare with Traditional Machine learning and what data needs to be collected for both types of approaches. • Planned to finish Research this week and create a Plan of Approach presentation for next Wednesday (14 Dec). • Before the 16th of December, collect sensor data for training an object detection model using various cameras to increase the robustness of the detection model. ◦ Planned to look into what kind of data is needed to create a detection model capable of identifying different scenarios the sensors could be in (upside-down, missing Matrix Code, rotated, etc.).	

14 Dec 2022	• Talked with Rogier about the Plan of Approach documentation ◦ Research Questions were missing from the PoA. ◦ A lot of details were put into the documentation, but there is not always a reason or motivation for choosing something for the approach, this should be added to future documentation • The final documentation will be no more than 50 pages, and the final presentation will be 20 minuets, with 30 minuets of questioning after. • Future meetings with Rogier will be every 4 weeks, and will hopefully line up with weekly meetings, but this will be determined closer to the meeting date. • As a communication plan, it was decided that sending the weekly meeting notes to Rogier would be beneficial for keeping everyone in the loop, and keep meetings well documented.	• Make a list of requirements for a solid goal of the project • Make research questions from these requirements and add them to the Plan of Approach
11 Jan 2023	• Talked with Fabian and Dieuwe about the Research Questions for the Plan of Approach Documentation and what changes should be made. • Presented the progress of the Cascade Model and what I was able to achieve since the last meeting. ◦ Talked about what approaches could help make the model more robust in the future and which issues of the model to tackle next.	

	• Presented the progress with the physical set up and what the next steps are with getting it working as an initial prototype for the final product.	
18 Jan 2023	• Covered the Research Questions so far and what changes should be made for the documentation. • Presented data augmentation and it's results on the training, as well as discussed annotation augmentation. • Talked about work done with the Raspberry Pi, and how it can be used moving forward. • Decided to focus the next week on the setup and image collection and preparing the current documentation for the Mid-Term Report.	• Wim Luyendijk Finish introduction to the documentation. • Wim Luyendijk Find out about the report template. • Wim Luyendijk Work on the 3D model for the camera mount. • Wim Luyendijk Include annotation augmentation. • Wim Luyendijk Fabian Girrbach Look at the camera lens focus. • Wim Luyendijk Fabian Girrbach Find documentation on how to SSH to the Raspberry Pi from windows. • Wim Luyendijk Begin collecting matrix code images with the Pi Camera. • Wim Luyendijk Add documentation on how to use the 3D printer if incomplete here [Student Version] Use of 3D printing explained
25 Jan 2023	• Talked about the mid-term documentation and getting the requirements needed for the final documentation and the topics of my next meeting with Rogier. • Discussed 3D printing and simplifying the model to avoid the need for fine tuning measurements. • Covered work done for the annotation Augmentation, currently the program can read annotated text files and translate rectangle positions, but work still needs to be done on proper alignment to the image translation.	• Wim Luyendijk Ensure the library is correct • Wim Luyendijk (best possible image) Camera in combination with the library is working (pre annotated images) Good weather situation works 100% • Wim Luyendijk Fix rectangle bugs • Wim Luyendijk Graph results with different variables (lighting, blur, rotation, brightness) • Wim Luyendijk meet with Rogier on the 27th

	• Showed a Demo of how the model reacts to images captured from the physical structure, and found a bug that may be causing matrix codes to not be able to read.	
01 Feb 2023	• Talked about the physical Structure and it's changes since the last meeting, specifically the change in lens attachment which improved the quality of the images. • Presented a Demo of how the library interacts with the new image quality and explained a bug from the library when a Matrix Code isn't readable. • Discussed the mid-term documentation and clarified what Movella does, and what the next steps are for the Final Documentation • For next week: Get current program to a stable version, and start working on the planning phase of the GUI,	• Wim Luyendijk Start planning GUI, frameworks and dataflow • Wim Luyendijk Flow of: Input image and get out it's data and position • Wim Luyendijk create Graphs showing accuracy for the object detection and library, (false positive rates, confusion matrix.)
08 Feb 2023	• Talked about the dataflow of the program and how this relates to the GUI and what architectural approaches could be taken for the GUI system. • Talked about the program so far and what tools can be used to keep the program up to standard. • Formatting, Auto Doc, yapf and pep8 and interface classes.	• Wim Luyendijk Prepare a presentation of the GUI approach to present during the next meeting. 15 Feb 2023 • Wim Luyendijk Create an overview of possible approaches regarding architectur and flow. • Wim Luyendijk Make wireframe diagrams of the program interface. • Wim Luyendijk Research the tool necessary for the preferred architectural approach. • Wim Luyendijk Plan a high level overview of the timeframe of the GUI tha includes milestones and tasks.

15 Feb 2023	• Presented GUI pages, Architectural approaches, time frame and tasks to Fabian and Dieuwe. • Talked about features of the application and what is needed for the system. • Planned for Scrum approach during GUI development with stand ups and weekly sprints • Planned meeting with operators to get feedback on the GUI ideas	• Wim Luyendijk meet with operators about GUI Frames • Wim Luyendijk Make a quick and simple overview of the project for operators • Wim Luyendijk Plan out Scrum per week to the end of the GUI development • Wim Luyendijk Start work GUI and loading images from PC • Wim Luyendijk Research how to connect PC and Pi
22 Feb 2023	• Presented the GUI layout so far and showed a demo of the navigation bar. • Talked about the first Sprint and what was expected in following a SCRUM approach.	• Wim Luyendijk Add basic functionality to the GUI • Wim Luyendijk Prepare backlog of issue for Sprint 2 • Wim Luyendijk start planning API for connecting the backend to frontend
03 Mar 2023	• Met with Dieuwe and presented the results of the first sprint • Talked about the what changes needed to be made for the next sprint • Covered tasks of sprint 2 and what issues should be addressed before the next meeting	
08 Mar 2023	• Met with Fabian and Dieuwe and showed a demo of the application after sprint 2 • Went over backlog of issues and which were the most important to focus on for sprint 3 • Discussed what approaches could be done to ensure the application meets the requirements of the project • Talked about what functionality is still needed from the application	• Wim Luyendijk Add interfaces and documentation to backend in order to connect Raspberry Pi to the application • Wim Luyendijk Get application workin as expected for a good example image • Wim Luyendijk Get application workin as expected for a bad example image

C. Back end Code

C.1 Object Detector

```
import cv2
import os

import methods.readMatrix as readMatrix
import methods.helper as helper


class MtiDetection:
    def __init__(self, x, y, w, h) -> None:
        self._topCoordinate = (x, y)
        self._width = w
        self._height = h
        self._deviceId = None
        self._labelId = None
        self._listId = None
        self._isRead = False
        self._croppedImage = None

    def getRectangle(self):
        rect = (self._topCoordinate, self._width, self._height)
        return rect

    def getCroppedImage(self, newImage, positionX, positionY, paddingY,
paddingX):

        cropped = newImage[positionY - int(paddingY / 4):positionY +
int(paddingY * 1.2),
                           positionX - int(paddingX / 4):positionX +
int(paddingX * 1.2)]
        return cropped

    def setImage(self, image):
        self._croppedImage = image

    def getXCoord(self) -> int:
        return self._topCoordinate[0]

    def getYCoord(self) -> int:
        return self._topCoordinate[1]

    def getIsRead(self):
        return self._isRead

    def setListId(self, id):
        self._listId = id

    def __str__(self):
        rect = (self._topCoordinate, self._width, self._height)
        print("printing: " + rect.__str__)


def findMatrixCodes(directory: str, cascade_Matrix):
    """findMatrixCodes(directory, cascade_Matrix)
    This function is used to scan a given image and find the
    Matrix Codes, each found code's position is then passed to getMatrixData
function

    Args:
        directory (str): The path to the folder of images you want to process.
        cascade_Matrix (xml file): The trained detection model used for
detecting Matrix Codes.
    """

    # Loading the image into the program
    image = cv2.imread(directory)  # + "/" + filename)
    copy = cv2.imread(directory)  # + "/" + filename)

    # Object Detection Model looking for Matrix Codes within the image
    rectangles = cascade_Matrix.detectMultiScale(image)
    rectangleFound = False

    sensorList = []
    for (x, y, w, h) in rectangles:

        # Filter out false detections
        if w > 70 and w < 100 and h > 70 and h < 100:
            testVar = MtiDetection(x, y, w, h)

            # saving the cropped image
            testVar._croppedImage = readMatrix.cropImage(image, x, y, w, h)

            sensorList.append(testVar)
    return sensorList

### End of Function ###


def labelImage(rectangles: list[MtiDetection], copy):
    image = cv2.imread(copy)

    for testVar in rectangles:
        x, y = testVar._topCoordinate
        w = testVar._width
        h = testVar._height
        if testVar._isRead:
            color = (255, 0, 0)
        else:
            color = (0, 0, 255)
        cv2.rectangle(image, (x, y), (x + w, y + h), color, 5)
        # get space needed for annotation of bounding box
        (w, h), _ = cv2.getTextSize(
            str(testVar._listId), cv2.FONT_HERSHEY_SIMPLEX, 1.4, 2)

        # print the text with background
        cv2.rectangle(image, (x, y-(h*2)), (x+w, y), color, -1)
        cv2.putText(image, str(testVar._listId), (x, y-10),
                    cv2.FONT_HERSHEY_SIMPLEX, 1.4, (255, 255, 255), 2)
    return image


def getCodeList(rectangles: list[MtiDetection], copy) -> list[MtiDetection]:
    image = cv2.imread(copy)
    data = []
    counter = 0
    for testVar in rectangles:
        x, y = testVar._topCoordinate
        w = testVar._width
        h = testVar._height
        testVar._deviceId = readMatrix.getMatrixData(
            image, x, y, int(w), int(h), "filename")
        if testVar._deviceId == "not read":
            testVar._isRead = False
        else:
            testVar._isRead = True
        counter += 1
        data.append(testVar)
    return data


def getBadList(dataList: list[MtiDetection]) -> list[MtiDetection]:
    badList = []

    for id in dataList:
        if not id._isRead:
            badList.append(id)

    return badList


def getX(data: MtiDetection) -> int:
    return (data._topCoordinate[0])


def getY(data: MtiDetection) -> int:
    return (data._topCoordinate[1])


def sortList(dataList: list[MtiDetection]) -> list[MtiDetection]:

    sortedList = []
    rowList = []
    rowSizeList = []
    rowNumber = 0

    # sort by row
    dataList.sort(key=getY)

    # Cluster detections by rows
    lastRectX = dataList[0].getYCoord()
    rowCounter = 0
    for data in dataList:
        if abs(int(lastRectX) - int(data.getYCoord())) > 60:
            # get the row size
            rowSizeList.append(rowCounter)
            rowCounter = 0
        lastRectX = data.getYCoord()
        rowCounter += 1
    # append last row of sensors
    rowSizeList.append(rowCounter)

    rowCounter = 0
    rowNumber = 0
    minRange = 0
    # for data in dataList:  # Sort through each row
    # range example: 0-4, 5-9, 10-14
    # columns per row change on false detections
    for dynamicRange in range(0, len(rowSizeList)):
        for row in range(minRange, minRange+rowSizeList[rowNumber]):
            if row < len(dataList):
                rowList.append(dataList[row])

        # sort row
        rowList.sort(key=getX)
        # Adding row to sorted list
        for item in rowList:
            sortedList.append(item)
        # Preparing list for next row
        rowList = []
        minRange = minRange + rowSizeList[rowNumber]
        rowNumber += 1

    # Print coords of sorted list
    listId = 0
    print("list of sorted coordinates: ")
    for item in sortedList:
        listId += 1
        item._listId = listId
        testVar = item
        print(str(testVar._topCoordinate) + ", ", end='')

    return sortedList
```

C.2 Matrix Reader

```
from pylibdmtx.pylibdmtx import decode

def getMatrixData(srcImage, positionX, positionY, paddingY, paddingX, lable):
    """getMatrixData(image, X, Y, paddingY, padding X, lable)
    This function is used to read the data from a matrix code that is provided
    by the object detection algorithm. The data is printed to the console and
    a boolean is returned stating if the Matrix Code could be read.

    Args:
        srcImage (image): Image to be processed by the datamatrix library.
        positionX (int): The X position of the detected Matrix Code.
        positionY (int): The Y position of the detected Matrix Code.
        paddingY (int): The width of the detected Matrix Code.
        paddingX (int): The Height of the detected Matrix Code.
        lable (str): The name of the image or sensor ID.

    Returns:
        bool: Retruns a boolean on whether or not the library could read the
Matrix Code within the image.
    """

    data = False

    # check the amount of pixels

    # Cropping the image to get a smaller scan space, this significantly
decreases runtime
    cropped = cropImage(srcImage, positionX, positionY, paddingX, paddingY)
    # image[y.start:y.end, x.start:x.end]

    # Commented code below sharpens the edges of Matrix Codes when the
resolution is low
    # by creating a threshold between black and white pixels.

    # Adding image manipulation for best readability for low res cameras
    # grayScale = cv2.cvtColor(cropped, cv2.COLOR_BGR2GRAY)
    # applying the adaptive gaussian threshold
    # ret, thresh = cv2.threshold(grayScale, 255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C| cv2.THRESH_BINARY, 11, 2)

    # Decode the Data Matrix
    decodedData = decode(cropped)
    # Check if the decoding worked for that Matrix Code,
    if len(decodedData) > 0:
        print("Matrix code: " + str(lable) + ", data: " +
              str(decodedData[0].data, "utf-8"))
        data = True
        return str(decodedData[0].data, "utf-8")
    else:
        print("Could not read the Data Matrix: " + str(lable))
        return "not read"
```

```
def cropImage(srcImage, positionX, positionY, paddingX, paddingY):
cropped = srcImage[positionY - int(paddingY / 4):positionY + int(paddingY *
1.2), positionX - int(paddingX / 4):positionX + int(paddingX * 1.2)]

    return cropped

### End of Function ###
```