ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΔΙΑΤΜΗΜΑΤΙΚΟ ΠΡΟΓΡΑΜΜΑ ΜΕΤΑΠΤΥΧΙΑΚΩΝ ΣΠΟΥΔΩΝ

«ΣΥΣΤΗΜΑΤΑ ΑΥΤΟΜΑΤΙΣΜΟΥ»

Μεταπτυχιακή Εργασία

Electromechanical design and assembly of an automated Quality Inspection automated station and implementation of ANN based defect detection

Ηλεκτρομηχανολογικός σχεδιασμός και κατασκευή αυτοματοποιημένου σταθμού Ελέγχου Ποιότητας και εφαρμογή ελέγχου ελαττωμάτων βασισμένου σε Τεχνητά Νευρωνικά Δίκτυα

Σπυρίδων Κανακάκης

Επιβλέπων Καθηγητής: Πανώριος Μπενάρδος

ΑΘΗΝΑ 2022

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Μηχανολόγων Μηχανικών,
Τομέας Τεχνολογίας των Κατεργασιών

Electromechanical design and assembly of an automated Quality Inspection automated station and implementation of ANN based defect detection

Ηλεκτρομηχανολογικός σχεδιασμός και κατασκευή αυτοματοποιημένου σταθμού Ελέγχου Ποιότητας και εφαρμογή ελέγχου ελαττωμάτων βασισμένου σε Τεχνητά Νευρωνικά Δίκτυα

Διπλωματική Εργασία

του

ΚΑΝΑΚΑΚΗ Δ. ΣΠΥΡΙΔΩΝΟΣ

στα πλαίσια του

Διατμηματικού Μεταπτυχιακού Προγράμματος Σπουδών «Συστήματα Αυτοματισμού»

Επιβλέπων : ΠΑΝΩΡΙΟΣ ΜΠΕΝΑΡΔΟΣ
Επίκουρος Καθηγητής

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την ……………………….

(Υπογραφή)	*(Υπογραφή)*	*(Υπογραφή)*
....	....	....

Αθήνα, Οκτώβριος 2022

Εθνικό Μετσόβιο Πολυτεχνείο
Σχολή Μηχανολόγων Μηχανικών,
Τομέας Τεχνολογίας των Κατεργασιών

ΔΗΛΩΣΗ ΜΗ ΛΟΓΟΚΛΟΠΗΣ ΚΑΙ ΑΝΑΛΗΨΗΣ ΠΡΟΣΩΠΙΚΗΣ ΕΥΘΥΝΗΣ

Με πλήρη επίγνωση των συνεπειών του νόμου περί πνευματικών δικαιωμάτων, δηλώνω ενυπογράφως ότι είμαι αποκλειστικός συγγραφέας της παρούσας Πτυχιακής Εργασίας, για την ολοκλήρωση της οποίας κάθε βοήθεια είναι πλήρως αναγνωρισμένη και αναφέρεται λεπτομερώς στην εργασία αυτή. Έχω αναφέρει πλήρως και με σαφείς αναφορές, όλες τις πηγές χρήσης δεδομένων, απόψεων, θέσεων και προτάσεων, ιδεών και λεκτικών αναφορών, είτε κατά κυριολεξία είτε βάσει επιστημονικής παράφρασης. Αναλαμβάνω την προσωπική και ατομική ευθύνη ότι σε περίπτωση αποτυχίας στην υλοποίηση των ανωτέρω δηλωθέντων στοιχείων, είμαι υπόλογος έναντι λογοκλοπής, γεγονός που σημαίνει αποτυχία στην Πτυχιακή μου Εργασία και κατά συνέπεια αποτυχία απόκτησης του Τίτλου Σπουδών, πέραν των λοιπών συνεπειών του νόμου περί πνευματικών δικαιωμάτων. Δηλώνω, συνεπώς, ότι αυτή η Πτυχιακή Εργασία προετοιμάστηκε και ολοκληρώθηκε από εμένα προσωπικά και αποκλειστικά και ότι, αναλαμβάνω πλήρως όλες τις συνέπειες του νόμου στην περίπτωση κατά την οποία αποδειχθεί, διαχρονικά, ότι η εργασία αυτή ή τμήμα της δεν μου ανήκει διότι είναι προϊόν λογοκλοπής άλλης πνευματικής ιδιοκτησίας.

(Υπογραφή)

. . . . . .

Κανακάκης Σπυρίδων

Περίληψη

Στόχος της παρούσας διπλωματικής εργασίας είναι ο ηλεκτρομηχανολογικός σχεδιασμός και η κατασκευή ενός αυτοματοποιημένου σταθμού Ελέγχου Ποιότητας και η εφαρμογή ελέγχου ελαττωμάτων βασισμένου σε τεχνητά νευρωνικά δικτύα. Το πρόβλημα που στοχεύει να επιλύσει το προτεινόμενο σύστημα είναι η ενσωμάτωση αυτοματοποιημένου συστήματος ποιοτικού ελέγχου, βασισμένου σε οπτική είσοδο δεδομένων, με μειωμένη πολυπλοκότητα στο σχεδιασμό και ενσωμάτωση σε μια γραμμή παραγωγής, λιγότερο τεχνικά απαιτητικό στην εφαρμογή του και σε κάποιο βαθμό οικονομικά προσιτό, σε σύγκριση με τα υπάρχοντα συστήματα που έχουν εφαρμοστεί σε γραμμές παραγωγής.
Ως προς την διαδικασία που ακολουθήθηκε, αρχικά, ολοκληρώθηκε ο μηχανολογικός σχεδιασμός του συστήματος, συμπεριλαμβανομένου του δομικού πλαισίου και των συστημάτων μετάδοσης κίνησης και συναρμολογήθηκαν όλα τα μηχανικά μέρη. Στη συνέχεια ολοκληρώθηκε ο ηλεκτρολογικός σχεδιασμός του αυτοματοποιημένου σταθμού, συμπεριλαμβανομένης της επιλογής κινητήρων, του σχεδιασμού του ηλεκτρικού κυκλώματος για τον έλεγχο του σταθμού και όλων των πρόσθετων ηλεκτρικών μερών και καλωδίων που απαιτούνται για τη σωστή λειτουργία του σταθμού. Στη συνέχεια ολοκληρώθηκε η συναρμολόγηση των ηλεκτρικών μερών επί του πλαισίου και των μηχανολογικών υποσυστημάτων.
Παράλληλα, αναπτύχθηκε το λογισμικό ελέγχου του αυτοματοποιημένου σταθμού. Αυτή η διαδικασία περιλάμβανε την ανάπτυξη του Γραφικού Περιβάλλοντος Διεπαφής Χρήστη, του λογισμικού ελέγχου κίνησης του συστήματος, συμπεριλαμβανομένης της τροφοδοσίας εικόνας από την κάμερας, ενός γραφήματος θέσης της κάμερας και ένα παράθυρο καταγραφής σφαλμάτων και, τέλος, τον προγραμματισμό του μικροελεγκτή, ο οποίος λειτουργεί μεσολαβεί μεταξύ του λογισμικού ελέγχου κίνησης και ορισμένων ηλεκτρολογικών μερών.
Αναπτύχθηκε επίσης ένα μοντέλο ελέγχου ελαττωμάτων που βασίζεται σε Τεχνητό Νευρωνικό Δίκτυο για την αναγνώριση ελαττωμάτων σε αντικείμενα από χυτό αλουμίνιο. Ο σκοπός αυτού του μοντέλου ήταν να επιβεβαιώσει τη λειτουργία του αυτοματοποιημένου σταθμού ως μέσο εφαρμογής ποιοτικού ελέγχου. Για την ανάπτυξή του, αρχικά επιλέχθηκε η αρχιτεκτονική του μοντέλου, ακολούθησε η επιλογή και η επισήμανση των εικόνων για ενός συνόλου εικόνων για την εκπαίδευση του μοντέλου και στη συνέχεια πραγματοποιήθηκε η εκπαίδευση του μοντέλου. Μετά την ολοκλήρωση της εκπαίδευσης, χρησιμοποιήθηκαν συγκεκριμένες μετρικές για την αξιολόγηση της απόδοσης του μοντέλου και παρουσιάστηκε η διαδικασία ενσωμάτωσής του στο λογισμικό ελέγχου της διάταξης.
Τέλος, εξετάστηκε η λειτουργικότητα της ηλεκτρομηχανολογικής διάταξης και του λογισμικού ελέγχου, καθώς και η απόδοση του εκπαιδευμένου μοντέλου Τεχνητού Νευρωνικού Δικτύου και έγιναν ορισμένες προτάσεις για τη βελτίωσή τους.

Λέξεις Κλειδιά

Αυτοματοποιημένος σταθμός, Έλεγχος Ποιότητάς, Έλεγχος Σφαλμάτων, Μηχανολογικός σχεδιασμός, μικροελεγκτής, Arduino, GRBL, Nema 17, Γραφικό Περιβάλλον Διεπαφής Χρήστη, Qt5, pyQt5, OpenCV, matplotlib, Τεχνητά Νευρωνικά Δίκτυα, Deep Learning, SSD, TensorFlow, TensorBoard

Abstract

The goal of this thesis is the electromechanical design and assembly of an automated Quality Inspection station and the implementation of Artificial Neural Network based defect detection. The problem the proposed system aims to resolve is the integration of automated Quality Inspection system based on optical input with decreased complexity in design and integration to a production line, less technically demanding in its implementation and to a certain degree affordable, compared to the existing systems found in production lines.

The mechanical design of the system was firstly, completed, including the structural frame and the motion translation systems and all mechanical parts were assembled. The electrical design of the automated station was then completed, including the selection of motors, the design of the electrical circuit for the control of the station and all additional electrical parts and wiring required for the proper function for the station. The assembly of the electrical parts to the mechanical frame and sub-assemblies was then completed.

Concurrently, the control software of the automated station was developed. This process included the development of the Graphic User Interface, the motion control software for the control of the system, including the camera feed, a position graph of the end effector and an error dialog and lastly, the programming of the microcontroller, which acts as an intermediate between the motion control software and certain individual electrical parts.

An Artificial Neural Network based defect detection model was also developed to recognize defects on aluminum cast items. The purpose of this model was to establish the function of the automated station as a Quality Inspection system. For its development, the model architecture was firstly chosen, followed by the selection and labelling of the images for a training dataset and then the training of the model. After the training was complete, certain metrics were utilized to evaluate the performance of the model and the process of its integration was presented.

Lastly, the functionalities of the electromechanical assembly and the control software were examined, as well as the performance of the trained Artificial Neural Network model, and certain suggestions were made for their improvement.

Keywords

Automated station, Quality Inspection, Defect Detection, Mechanical Assembly, Arduino, GRBL, Nema 17, Graphic User Interface, Qt5, pyQt5, OpenCV, matplotlib, Artificial Neural Networks, Deep Learning, SSD, TensorFlow, TensorBoard

Ευχαριστίες

Η εκπόνηση και συγγραφή της παρούσας διπλωματικής εργασίας δεν θα μπορούσε να πραγματοποιηθεί χωρίς την αμέριστη στήριξη του επιβλέποντα Επίκουρου Καθηγητή κου. Πανώριου Μπενάρδου. Η καθοδήγησή του στα διαδικαστικά και επιστημονικά ζητήματα που πραγματεύεται η παρούσα εργασία υπήρξε πολύτιμη, όπως επίσης και η κατανόηση και υπομονή που επέδειξε στις δυσκολίες που προέκυψαν κατά την εκπόνηση της παρούσας.

Table of Contents

Table of Figures

1. Introduction

1.1 Background

Reliable and accurate quality control is an important element in industrial manufacturing. The repetitive nature of the processes involved in quality control has resulted in the field being one of the most automated, within the manufacturing world. However, with any significant increase in the complexity of the product subjected to quality control or the characteristics subjected to inspection, the overall processes become exponentially more complex for an automated system to perform.

Manufacturing goods in general and more specifically metal casting, which will be examined within the scope of this thesis, are subjected to quality control in search of certain characteristic that divert from the required end product (Venkat Sai, 2017). One such characteristic, specifically for cast items, is dimensional accuracy and is generally the easiest to be automated with simple electromechanical additions within a production line. Another one regards to the presence of defects in the surface of a product or a part of a product, such as pinholes or cracks. One other kind of quality control regards the intended functionality of the final product and can vary immensely between different products. As mentioned, the detection of first kind of defect, the dimensional, has already been automated to a great degree as a result of the rapid progress of manufacturing lines in the past century. The detection for the latter two kinds has also been automated to quite an extensive degree, especially during the past three decades, as computing technologies have provided way to implement automation and cease to rely purely on the presence of human experts.

In respect to the detection of defects in manufactured products, the most utilized technology is computer vision. There are several factors that make it an ideal technology for such application. The most prominent would have to be its seemingly intuitive nature. Defect detection at any product has been traditionally performed visually by highly competent individuals that were very well versed to the manufacturing processes of a product and as such could provide an accurate assessment on the existence of a defect and perhaps even the source of the defect within the manufacturing line. The cumulative knowledge of these individuals could not be utilized in the automation of these processes, if the transition was not at some degree intuitive to the individuals themselves, which at the very least means visual in nature. This is highlighted because there are numerous other defect detection methodologies not visual in nature such as laser scanning, ultrasound etc. that are not as intuitive as the purely visual observation.

The rapid advancement of Computer Vision (CV) during the past years has allowed for the implementation of such control in manufacturing lines (Anand, Sheila, 2019). Even without any quality inspection automation included in computer vision, an expert can assess the quality of a product without being physically present in the production line, which at the very least improves the working conditions of the individual and possibly increases the speed of assessment.

Concurrently, the progress of object detection algorithms has been tremendous the past twenty years (Zhong-Qiu Zhao et. al., 2019). These algorithms, which will be thoroughly explained in a later chapter, belong to the field of Artificial Intelligence (AI) and more specifically the field of Machine Learning (ML). The latter field has experienced astonishing progress within the last decade, both regarding its capabilities and the ease by which it can

be implemented, despite its high complexity. It is, thus, far from novel to see quality control being implemented in manufacturing lines, with ML algorithms performing tasks, otherwise performed by human experts, utilizing CV implementations.

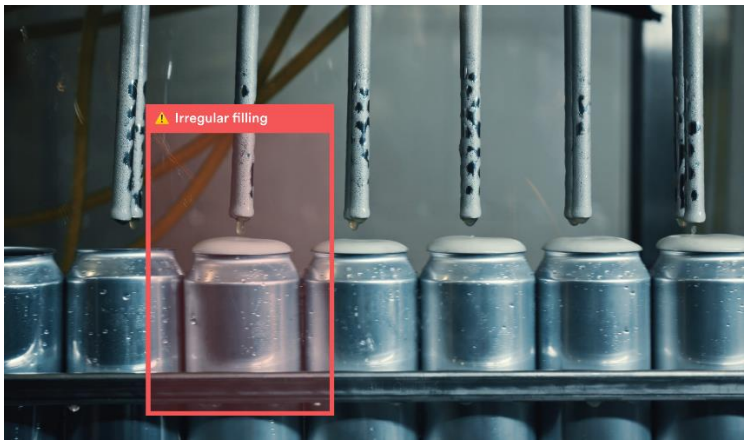


Figure 1: Iimplementation of Computer Vision based Quality Inspection

These systems usually consist of high-quality cameras, fixed on a specific point in the manufacturing line, feeding images, either continuously in live feed or in still frames, to a computer which runs an object detection algorithm (Anand, Sheila, 2019). Upon detection of a defect, the item is separated from the rest of the manufactured goods, either by labelling or simply by being singled out from the line with some form of actuation. Such implementations can also occur for packing and shipping of goods as well, or even at a designated quality control stage during production.

Figure 2: Integration of high-quality cameras for quality inspection.

These ML-enabled quality control systems, due to the complexity of these productions lines as well as the complexity of the systems themselves, are in most cases custom built to fit a specific production line of a specific product. Implementation of the hardware and more so, the software of these systems, is a demanding task that requires significant resources in the form of hardware specifications, highly skilled personnel and time required for the development of custom object detection algorithms. As it will be thoroughly explained further on, slight differences on the inputs of these algorithms or insufficient resources available can render these algorithms insufficient in terms of speed and accuracy of prediction to the point where their implementation cannot be justified.

Even if the resources are available and the implementation of a ML model is successful in the quality control of a production item, any alteration in the conditions of the prediction can render the model extremely insufficient. There are steps that can ensure the effectiveness of a model within a much greater range of conditions, but these steps require a great level of technical knowledge and in some cases even greater resources. It should be noted that the immense progress in the field of ML the past decade has led to the creation of certain development frameworks (Goldsborough, 2016) that make this process much less complicated at a development level, however, proper utilization and optimization of these algorithms still remains a highly technical matter.

In respect to the hardware, the requirements for its specifications vary depending on the product and the production line. Cameras are in most cases implemented within the production line on or near existing machinery equipment. Video or image feed is usually transmitted to a nearby computing unit, which runs the ML algorithm that detect defects. The computer, upon detection of a defect, responds in a pre-programmed manner. The cameras used are usually of high quality of image and at a high enough frames-per-second to keep up with the speed of the production line and their positioning is relevant to the stage of the production line in which defect detection is implemented and is determined by each application.

The result of this high degree of complexity, the hardware selection and specification, the highly technical nature of ML development, integration, optimization and generalization and the resulting resources required, both in time and in cost (Jianglin Huang, 2015), for the overall systems, have been the reason why these automated quality control systems have been used almost exclusively in high output production lines. For any other application, with a smaller output, the costs for the integration of such systems usually rarely outweigh the benefits, even in a long enough time frame.

1.2 Problem Statement

The limiting factors described in the previous chapter, impede the application of new developments in the field of ML in the automation of quality control for low production output. At this time, there are no commercial products available for integration in a production environment that can accomplish this task, that include both hardware and software.

Software frameworks, tools and solutions that enable the use of computer vision and ML in quality control operations do exist, however these do not come in the form of a product that integrates all the required hardware with the software. Furthermore, the software mentioned either requires a certain degree of coding experience, ML knowledge and hardware integration skills or in most cases is costly.

A demand therefore rises for the development of an integrated solution of ML integrated quality control for low production outputs that is of decreased complexity in its design and integration to a production line, less technically demanding to implement and to a certain degree affordable.

The proposed solution, which is the subject of the current thesis, is the electromechanical design and assembly of an automated station that can control the movement of a high-quality

camera within its working area, in order to capture images of products, which will then be subjected to defect detection.

The electromechanical design of this station should be robust enough to ensure proper function and capability of integration to a variety of production environments, yet with materials, both mechanical and electrical, that allow for a simplified and cost-effective construction, therefore, either commercially available or easily manufactured and assembled.

For the control of the automated station, a control software will be developed, including both the necessary programming for the function of the station and the graphic user interface (GUI) for the use of the station by a user. The steps and tools for the development of the control software will be presented and its design will be such that it allows the overall system to operate for a wide range of products and applications without any need for modification.

Furthermore, within the scope of this thesis, the advancements of ML and more specifically Artificial Neural Networks (ANNs), will be utilized in order to create a functional defect detection model. The theoretical background necessary for the development and optimization of this model will be thoroughly explained and the steps for the creation, evaluation, optimization and integration of the model will be illustrated.

The current thesis will conclude with an assessment of the functionality and robustness of the mechanical and electrical design and assembly of the automated station, the capabilities of the control software and the precision and speed of the defect detection model, with suggestions on the improvements of any possible issues that may arise for any of the above.

2. Mechanical design of automated station

The overall mechanical design of the automated station was based on the structural frame designed and assembled by Ms. Clara Berger, during her student internship at NTUA and as described in her report (Berger, 2018). The geometry of the frame has influenced the design process and selection of all other mechanical components. However, during the design process of the automated station, certain changes were deemed necessary both for structural reasons, as well as for the proper operation of certain subsystems and overall, the automated station.

In this chapter, the structural frame will be introduced, as it was previously designed and assembled, the changes that were deemed necessary will be highlighted, the actuation for the 3-axis camera module will be analyzed and the overall assembly of the mechanical components will be thoroughly described.

2.1Design of structural frame

The original frame was constructed with commercially available aluminum extrusion that can be easily cut at desired length and assembled into lightweight, often complex constructions. These extrusions are usually named after the dimensions of their cross section in millimeters, with the extrusion used in the original frame being a 3030-aluminum extrusion, meaning 30mm by 30mm in width and height in its cross section.

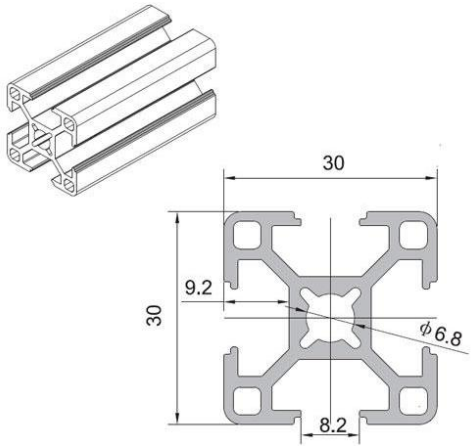


Figure 3: 3030 Aluminum extrusion with dimensions in mm

The assembly was possible with the use of 90-degree aluminum corner brackets placed at the junction of two aluminum extrusions. The fastening of these brackets was possible without any need for drilling with the use of specially designed nuts (tee-nuts) that slide in the ridges of the aluminum extrusion and when tightened are lodged firmly between the folds of each ridge. By combining aluminum extrusions of various lengths with the aforementioned parts, the structural frame was assembled. The same process was implemented for the construction of the mobile sub-assemblies of the frame.

Following the nomenclature of CNC systems (Smid, 2000) of equivalent modus operandi, the immobile, structural part of the frame represents the X- axis, and will henceforth be mentioned as such. The mobile parts of the frame, again by the same nomenclature, will be assigned as Y-axis for the horizontal mobile part and Z-axis for the vertical mobile part.

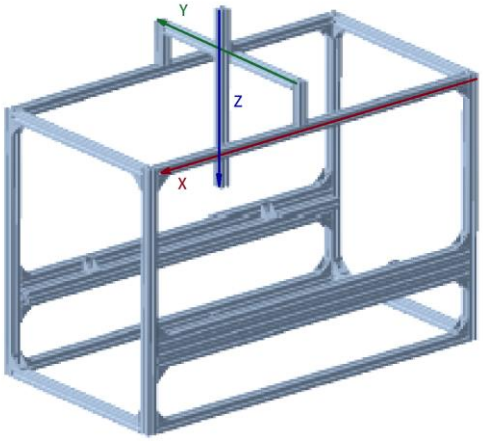


Figure 4: Final assembly of previous design

The mobility of the aforementioned parts was possible with 3D printed custom designed parts. These parts were inserted in the ridges of the aluminum extrusions, with tolerances that allowed the free slide of the 3D printed part along the ridge. Holes placed on the upper part of each 3D printed part allowed for the joining of each mobile sub-assembly, thus granting them freedom of movement along the length of the corresponding aluminum extrusion on which the 3D printed part was inserted.

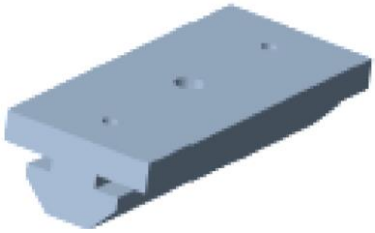


Figure 5: Design of 3D printed part for motion translation

Without any actuation present, both mobile parts were tested manually and they performed their intended purpose. However, upon the assembly of additional electromechanical parts, a significant issue arose. The Z-axis, meaning the structural part, could easily slide along the Y-axis without any equipment added to it. With the addition of all necessary equipment for the mobility of the camera module, such as the motors, transmission, mechanical parts etc., there was a significant increase in the weight of the Z-axis sub-assembly. The center of mass of the sub-assembly being further away from the center line of the aluminum extrusion and by extension, the 3D printed part inserted in each central ridge that allowed for the mobility of the Z-axis sub-assembly, meant that an equally significant torque was placed on the 3D printed part, that pushed it to be in contact with the outer surface of the aluminum extrusion closest to the Z-axis sub-assembly and the furthest inner surface of the ridge of the aluminum extrusion. The free body diagram of figure 6 clearly illustrates the issue:

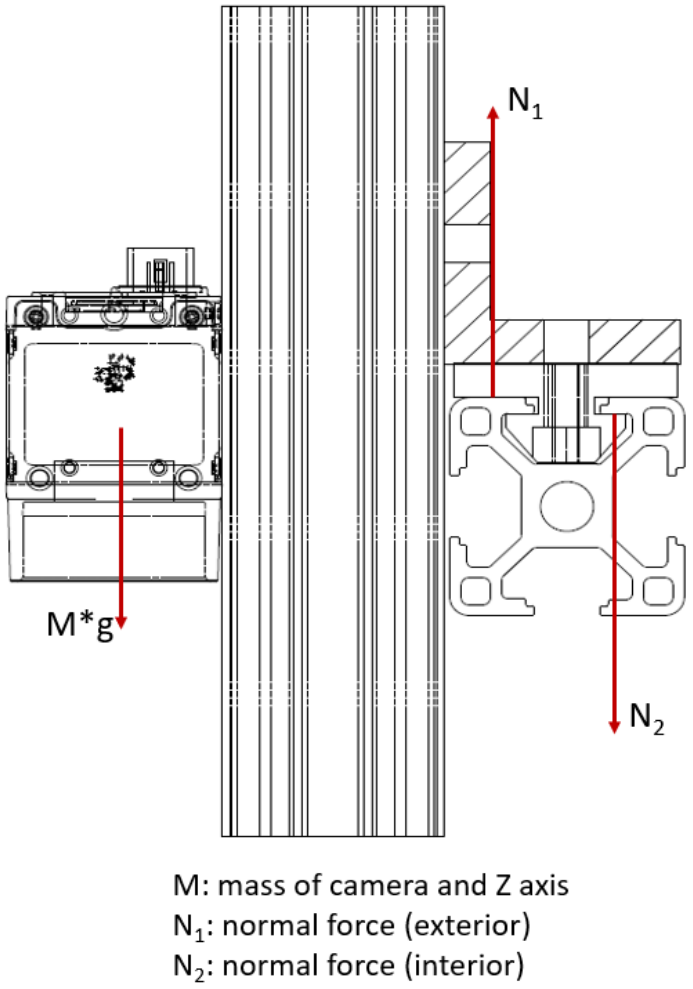


Figure 6: Freebody diagram of z-axis sub-assembly

The increased contact of the 3D printed part with the ridge, and the amount of force it placed on it, did not create an issue with the structural integrity of the sub-assembly or the 3D printed part. It did however increase the friction between the 3D printed part and the ridge, hence restricting the movement of the Z-axis sub-assembly. Additionally, the elasticity of the 3D printed part along with the loose tolerances that were chosen for the 3D printed part to allow for its unrestricted movement, meant that the torque introduced by the weight of the Z-axis sub-assembly was tilting the vertical axis of the Z-axis by a significant amount, causing loss of perpendicularity between the Z and Y axis of motion.

As an initial step, the tilting of the Z-axis was disregarded, as certain adjustments on the design of the mechanical components needed for the actuation of the camera module could allow for the vertical movement of the camera, regardless of the relatively minor alignment issues of the Z-axis. Calculations were made for the effect of the normal forces of the 3D printed part on the surface and the ridges of the aluminum extrusion, accounting for the final weight of the sub-assembly as it was preliminarily designed at that point and the friction they produced. The effect of the normal forces was enough to certainly cause issues with the 3D printed part overtime and the effect of the friction was not insignificant and would certainly increase the load on the motors. Furthermore, the effect of the friction was unilateral along the horizontal axis on which the Z-axis sub-assembly moved, which introduced torque that caused severe deviation from the perpendicularity between the Z and Y axis movements of

the two sub-assemblies.

The motor assigned to the Y-axis motion sub-system will have to accommodate the friction created by the movement of the whole Z-axis sub-assembly along the Y-axis, as illustrated in Figure 6. Though a specific motor for Z-axis has not been chosen at this point for this application, an assumption of a motor size is possible, given the overall size and capabilities of the individual components and the overall system, which would equate to a motor of a weight of 250gr. The calculation of the friction requires the normal force applied to the aluminum extrusion outer surface and inner ridge as a result of the torque created by the off-center mass of the Z-axis assembly. This torque is equal to the mass of the whole Z-axis sub-assembly multiplied by the distance of the center of gravity of the sub-assembly to the midline of the aluminum extrusion, which is the centerline of motion transfer of the Y axis motor. Both the mass and the distance from the midline can be easily calculated by the CAD design. The sum of the torques and the forces in considered zero, for the midline of the aluminum extrusion. By calculating these two normal forces with the principals laid above, the only data required is the coefficient of friction between a 3D printed part made from PLA and an aluminum extrusion.

$$T_{M_z} = M_z * g * l_{ZY} \ (1)$$
$$\Sigma(T) = 0 \ \rightarrow T_{M_z} = N_1 * 0.015 + 0.015 * N_2 \ (2)$$
$$\Sigma(F) = 0 \ \rightarrow N_1 = N_2 \ (3)$$

Where:
M_z: the mass of the whole Z-axis sub-assembly, here equal to 0.89kg
l_{ZY}: the distance of the center of mass of the Z-axis sub-assembly to the midline of the extrusion, here equal to approx. 32mm

Solving the system of the equations (1), (2), (3) in respect to T_{M_Z}, N_1 & N_2 , the resulting normal forces are the following:

$$N_1 = N_2 = 9.49N$$

Explicit bibliographical data were not found for the static and dynamic friction coefficient between anodized aluminum and 3D printed PLA. Since the very 3D printing process of the PLA can alter its geometry, a testing rig would have to be created and a series of test for the determination of the static and dynamic friction between specific 3D printed parts and anodized aluminum extrusion would be required. However, the setup of such a testing rig is well beyond the scope of this thesis as it is quite an elaborate and time-consuming process.

From existing resources, the sliding friction behavior of thermoplastics on aluminum vary from 0.22 to 0.45 depending on the sliding distance (Hechtel, 2021). The fact that PLA is a thermoplastic material (Van der Walt et. al., 2019) and anodized aluminum exhibits better tribological properties than untreated aluminum (Atraszkiewicz et. al., 2020), means that these data can be used for this application, with an added safety factor being the use of the upper values of the friction coefficient. So, in this regard, the coefficient factor will be chosen as 0.45.

From the equation of friction, the overall static friction resulting from the normal forces N_1 & N_2 is calculated, for a friction coefficient equal to $\mu = 0{,}45$, as:

$$F_{friction} = (N_1 + N_2) * \mu = 8{,}55N \ (4)$$

This force, the $F_{friction}$, is the load the motor for the Y-axis will have to overcome to initiate

the motion of the Z-axis sub-assembly along the Y-axis and it is more than significant enough to require a change in the mechanical design of the Y sub-assembly.

A complete redesign of the way the Z-axis sub-assembly was attached to the horizontal aluminum extrusion of the Y-axis sub-assembly was deemed necessary. The new design consists of two precision shafts that guide two linear bearings, both of which attach to the Z-axis sub-assembly, thus allowing it to move freely along the Y-axis horizontal distance. The precision shafts have been chosen with a diameter of 8mm, which is sufficient enough to withstand the eccentric load of the Z-axis sub-assembly, without introducing any visible deviations to the horizontal path along the Y-axis. Concurrently, the linear bearings offer minimal friction during the movement of the sub-assembly. This design is thoroughly analyzed in respect to the friction forces generated, on a latter chapter regarding the selection of motors.

The finalized design of the structural frame of the automated station is illustrated in the CAD design of figure 7.

Figure 7: Final 3D CAD design

2.2 Design of motion translation system

For systems with the main objective of allowing 3-axis mobility to a module, whether it being a spindle, a 3D printer head or any other tool, there are two prevalent motion translation methods. One is belt drive, which can be implemented in a number of different arrangements, most notably cartesian, coreXY and Hbot (Amridesvar et al., 2020). The other one is a translation screw, also known as power screw, with its most notable implementation being the lead screw and the ball screw. (Kaiji Sato et al., 1995) (Baluta, 2007)

Both of these translation methods are commonly implemented in all sorts of different mechanical systems and their modus operandi is considered common knowledge among the mechanical engineering domain. As such, for the purposes of this thesis, only the inherent differences between these two systems that directly affect the main objective of the automated station will be presented and compared.

Firstly, belt drive systems utilize a timing belt, usually teethed, that locks into toothed pulleys, thus transferring power usually from a motor to a moving part, via a series of arranged pulleys. These systems are preferred in high speed, high efficiency applications,

where the moving parts are usually relatively lightweight and there are no great demands for increased positional accuracy and repeatability. The reason for this is the presence of a timing pulley, which is usually constructed from an elastomer with significantly greater elasticity than steel. This elasticity is the root cause of most of its drawbacks such as the lower positional accuracy and repeatability, backlash effect, rippling effect during sudden changes in momentum, high degree of material aging, need for frequent re-tensioning and slipping under high load.

Translation screw systems use, as the name implies, a screw that via a nut translates rotational motion to linear. Depending on the nut and the geometry of the teeth on the screw there are various implementations. The most prevalent are the lead screw, with its variations, and the ball screw. The ball screw, most commonly found in high load, high precision applications is by far the most suitable solution for any system similar to its mechanics to the one studied in the present thesis. However, due to its high cost it cannot possibly be considered for this application.

Lead screw systems that are commonly available are the square thread and trapezoidal or acme thread. The former is reserved for high power transmission, which combined with its significant cost make it unsuitable for this application. The latter, most commonly used due to its versatility, low cost and ease of implementation has, as the name suggests, teeth of trapezoidal geometry. In comparison to the belt drive systems analyzed above, a trapezoidal lead screw system is much more robust under load or sudden changes in momentum, does not slip unless the maximum torque of the motor is exceeded in counter-torque, needs minimal maintenance to operate and is much easier to implement, with minimal mechanical parts added to the structural frame. The only drawback that is worth mentioning is the relatively lower speed compared to a belt drive system, with the difference between the two systems in speed not being relevant to the main objective of the automated station being designed.

In conclusion, it is apparent that the most suitable system to be implemented in the automated station is the trapezoidal lead screw. With this, we can proceed to the design of such a system. The main components are the trapezoidal screw, the nut, a bearing to support the weight of the screw on its free end and a motor coupler to attach the screw to the motor on the opposing end. Additionally, these components should be mounted on the structural frame.

The lead screw is placed parallel to the aluminum extrusion (or the precision shafts in the case of Y-axis), on the central axis of which a mobile part moves along. The lead screw attaches to the frame on one end via the bearing which is housed on a 3D printed part that is fastened to the frame. On the other end, as mentioned the lead screw attaches to the motor via the motor coupler. The motor is secured on the frame via an aluminum part that is fastened on the frame with tee-nuts.

Figure 8: Lead screw assembly

The nut is placed on the lead screw before the lead screw is secured on the frame, and it is fastened on a 3D printed part that acts as an intermediate connection between the nut and the mobile part the nut is transferring motion to. The connection is again made possible with bolts between the nut and the 3D printed part and the 3D printed part and the relative mobile part.

Following this implementation process and designing all the necessary intermediate parts, while taking into account the geometry of the structural frame with the changes presented in the previous chapter, the resulting motion translation systems for each axis are illustrated on figure 9.

Figure 9: Motion translation system for X, Y and Z axis

It should be noted, that by design the automated station cannot operate with a single translation system along the X-axis. Positioning only one such system unilaterally would introduce a torque that would rotate the 3D printed sliding parts within the range of their tolerances to be in contact with the corresponding aluminum extrusion ridges. As such, the 3D printed sliders would get stuck and the system would be motionless along the X-axis. To resolve that, there should be two motion translation systems along the X-axis, one for each of the horizontal aluminum extrusion it consists of, within which the 3D printed sliders reside. The motion of these two translational systems should be perfectly coordinated in order to prevent misalignment of the 3D sliders that will eventually cause them to be blocked within the ridges of the aluminum extrusion. This coordination can be achieved mechanically, but it requires elaborate design and introduces many challenges. It can be resolved however, during the electrical and software design latter on.

The 3D printed parts mentioned above, with the exception of the sliders, were developed firstly on a CAD program, within an assembly design of the entire electromechanical system. Once their dimensions where chosen to ensure proper function of the motion translation system and no interference with any other part of the assembly, the CAD file of the parts were exported as .STL files and 3D printed in a Raise 3D Pro 2 FDM (fused deposition modelling) 3D printer with PLA (polylactic acid) filament. The technical drawings of these parts can be found in Annex A.

2.3 Assembly of mechanical parts

With the design for the sub-assemblies for each of the three axes and the overall structural

frame, the finalized mechanical assembly is mostly complete and is illustrated in figure 10.

Figure 10: Final mechanical assembly

It should be noted that certain mechanical parts were also placed for provisional purposes. The overall mechanical design as originally designed and assembled and modified for the purposes of this thesis in the ways previously analyzed, is more than capable of responding to the demands of the main objective of the automated station. The addition of 90-degree aluminum corners to all inside corners of the structural frame was deemed necessary to allow for increased structural rigidity of the frame. Since, the frame consists of individual parts fastened with bolts and nuts and the operation of the automated station requires constant movement within this frame, inevitably these fastened connections would periodically require re-fastening due to the dynamic load from the acceleration and deceleration of the various moving parts. With the introduction of further supporting mechanical parts on the structural frame, the dynamic load is more evenly dispersed in a greater number of connections, thus the need for re-fastening can be greatly reduced or at the very least delayed significantly.

3. Electrical design of automated station

The electrical design of the automated station consists of all the non-mechanical parts of the overall assembly that allow for the automated operation of the individual axis, including motors, controllers, processing units, camera, sensors, power supply unit and of course all the necessary plugs and wiring.

Certain design parameters of the electrical assembly were influenced by the mechanical design of the station analyzed in the previous chapter. Concurrently, certain design decisions implemented during the mechanical design were made with regards to specific electrical components in mind. Such a cyclic process is necessary during the design of complex electromechanical systems as there are differences among both electrical and mechanical components, which otherwise produce the same results, that heavily alter the geometry of the final assembly. As such, it is inherently impossible or at the very least impractical to design either the mechanical or the electrical part of an assembly without taking into account the other.

With this clarification, the electrical components deemed most fit for this application are presented in the following chapters and the specifications for each component are illustrated.

3.1 Selection of electric motors

There are a lot of different kinds of electric motors that could potentially be suitable for implementation in an automated system. The specific needs of this application, the automated station, immediately reduces the multitude of different choices. Further analysis on the advantages and disadvantages of each of the different kinds of motors, in the context of the main objective of the automated station, further reduces the suitable choices.

Initially, the power demands for each motor should be specified. For the X and Y axis, there is only the dynamic load of the movement of the parts each axis is carrying. In this regard, the only parameter that needs calculating is the force of friction that the motors need to overcome to make that movement possible, which in this case is quite apparent that it is minimal. For the Z axis, the power demands are slightly more complex, as there is a permanent static load the motors are required to uphold, which is the weight of the camera module, along with the parts that transfer movement to the module, in addition to the dynamic load that results from the requirement to move the module along the Z-axis.

The results of the power calculations indicate that high power output motors are unnecessary in this application, which immediately excludes the use of any AC motor. The available options are therefore limited to relatively low powered DC motors. These include brushed and brushless DC motors, servo motors and stepper motors.

Figure 11: Brushed and brushless dc motor

Brushed motors operate on an on/off pattern, where they rotate when voltage is provided and are still when it is not. Though simple in their operation, they are inherently unsuitable for use cases where positioning is integral as they lack any form of integrated positioning control. A closed loop system for their control could be designed but it would increase the complexity of the system unnecessarily. The same restrictions apply to brushless motors. Additionally, both these types of motors, though more than capable in regards to power output, usually operate in high rotational speeds. This introduces further complexity to the systems, as gear reduction should be implemented to take advantage of the high-power output in a much more controlled rotational speed. These motors are quite common in robotic applications when closed loop control and gear reduction are implemented, but they are deemed unnecessary for this particular application.

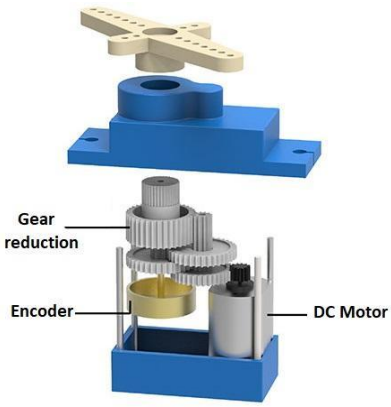


Figure 12: Servo motor – Operating principal diagram

Servo motors address most of the issues that arise from the DC motors. They are actually more of a rotational actuator, rather than simply a motor. They are composed of a DC motor, usually a gear reduction system and a sensor that provides positional feedback. They require a specific controller to operate, which reads the values of the position sensor and adjusts the voltage input of the DC motor, allowing for control of the angle and the velocity of the motor. These systems provide excellent precision and high performance and are favored in robotic and automation applications. However, such performance comes with the necessity for a rather complex construction of the motor and intricate control implementation by the controller, resulting in high costs. Lower cost servo motors exist but they come with a low torque output, with the cost increasing rapidly for any significant increase in torque output.

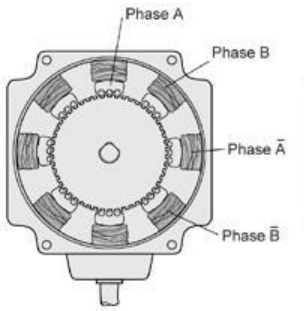


Figure 13: Stepper motor – operating principal diagram

Lastly, there are stepper motors. These motors operate in a completely different principle than common DC motors. They are brushless and their rotation is an addition of individual equal steps on the circumference of the rotor. They have inherent position control without

feedback (open loop), as by default they are operated step by step and can be controlled to hold each step. They are sized by the maximum opposing torque they can withstand while maintaining a step. These steps can increase in accuracy with a controlling technique called microstepping (Baluta, 2007) and even affordable ones can easily be micro-stepped to up to 1/32nd of their step, which results in significantly high accuracy. They are generally low cost, especially comparing servo motors of equal torque output. For all these reasons, stepper motors are favored for prototyping applications and are deemed most suitable for use on the automated station.

Figure 14: Nema 17 stepper motor

The load these motors are required to operate under can be calculated, for each axis and motor with the following equations:

- Z-axis:
  The motor assigned to the Z axis motion sub-system will only have to accommodate the weight of the camera and the 3D printed part connecting the camera to the motion sub-system. The combined weight is measured as $M = 0.182kg$. The torque required to raise this load in a lead screw is derived by the equation below (VCalc, 2022):

$$T_R = F \cdot \frac{d_m}{2} \cdot \frac{L+\pi \cdot \mu \cdot d_m}{\pi \cdot d_m - \mu \cdot L} \quad (5)$$

  Where:
  T_R: torque required to raise the load
  F: force opposed to the movement (the load), here equal to $M * g = 1.82N$
  d_m: mean diameter of lead screw, here equal to 8mm
  L: the lead, meaning the travel of the nut per one screw revolution, here equal to 2mm
  μ: the coefficient of friction between the brass nut and the steel lead screw with no lubrication added, here equal to 0.19 (VCalc, 2022)

  The result of the equation above is a required torque equal to 0.2N*cm. The unit of measurement being expressed as N*cm is such due to its commonality in datasheets of motors for such uses.

- Y-axis:
  The calculation of the load the motor of the Y axis has to overcome has already been illustrated on an earlier chapter, though for a different motion translation system and overall mechanical design. The load of the current mechanical implementation is the weight of the Z-axis sub-assembly, multiplied by the distance from the center of gravity of the overall sub-assembly to the center of the precision rods utilized in the

Y-axis sub-assembly. This torque T_{M_Z} results in two normal forces created, N_1 & N_2, one between each linear bearing and the corresponding precision rod.

$$T_{M_z} = M_z * g * l_{ZY} \ (6)$$
$$\Sigma(T) = 0 \ \rightarrow T_{M_z} = N_1 * l_N + N_2 * l_N \ (7)$$
$$\Sigma(F) = 0 \ \rightarrow N_1 = N_2 \ (8)$$

Where:
M_z: the mass of the whole Z-axis sub-assembly, here equal to 0.89kg
l_{ZY}: the distance of the center of mass of the Z-axis sub-assembly to the midpoint between the centers of the precision rods, here equal to approx. 37mm
l_N: the distance between the center of each precision rod and the midpoint between the two rods, here equal to 21.5mm

Solving the system of the equations 1, 2, 3 in respect to T_{M_Z}, N_1 & N_2 , the resulting normal forces are the following:

$$N_1 = N_2 = 7.84N$$

The effect of friction for these two normal forces can easily be calculated, given the friction of coefficient for the linear bearing chosen, which is 0.003 (Euro-bearings.com).

$$F_{friction} = (N_1 + N_2) * \mu = 0.047N \ (9)$$

And with the torque is calculated by the equation (10) as:

$$\mathrm{T_R} = \mathrm{F} \cdot \frac{\mathrm{dm}}{2} \cdot \frac{\mathrm{L} + \pi \cdot \mu \cdot \mathrm{dm}}{\pi \cdot \mathrm{dm} - \mu \cdot \mathrm{L}} = 0.02N * cm \ (10)$$

Where:
T_R: torque required
F: force opposed to the movement (the load), here equal to $F_{friction} = 0.0235N$
d_m: mean diameter of lead screw, here equal to 8mm
L: the lead, meaning the travel of the nut per one screw revolution, here equal to 2mm
μ: the coefficient of friction between the brass nut and the steel lead screw with no lubrication added, here equal to 0.19 (VCalc, 2022)

- X-axis

The X-axis motors have to overcome the load of the friction created by the combined masses of the Z and Y axis sub-assemblies, created between the 3D printed sliders and the anodized aluminum surfaces of the extrusions. Two normal forces, N_1 and N_2 are created at each of the two sliders of the X-axis sub-system, from the weight of the Z and Y sub-assemblies, which is easily calculated from the CAD file of the overall assembly. Each of these normal forces creates a friction, with a friction coefficient $\mu = 0.45$, as explained in chapter 2.1. Each of the two motors will have to overcome each of these two friction forces. The calculations for all the above are illustrated in the equations (12) – (14):

$$(M_Z + M_Y) * g = N_1 + N_2 \ (12)$$
$$N_1 = N_2 \ (13)$$
$$F_1 = F_2 = N_1 * \mu = N_2 * \mu \ (14)$$

With $M_Z = 0.89kg$ and $M_Y = 1.33kg$, $F_1 = F_2 = 24.6N$

And utilizing the torque equation for a lead screw motion translation system, the required torque for each motor is calculated by equation (15) as:

$$T_R = F \cdot \frac{dm}{2} \cdot \frac{L + \pi \cdot \mu \cdot dm}{\pi \cdot dm - \mu \cdot L} = 2.8N * cm \quad (15)$$

Where:
T_R: torque required
F: force opposed to the movement (the load), here equal to $F_{friction} = 24.6N$
d_m: mean diameter of lead screw, here equal to 8mm
L: the lead, meaning the travel of the nut per one screw revolution, here equal to 2mm
μ: the coefficient of friction between the brass nut and the steel lead screw with no lubrication added, here equal to 0.19 (VCalc, 2022)

In regards to the sizing of the stepper motor chosen, given the results from the calculation illustrated above on the motor torque demands of the axis sub-assemblies, the most suitable commercially available choice is the Nema 17 stepper motor, which is the industry standard for most 3D printers in production at the moment. The specifications for the Nema 17 motor (Schneider Electronics) are presented in figure 14, below.

Specifications

1.5 Amp motors		Single length	Double length	Triple length
Part number		**M-1713-1.5** • *(1)*	**M-1715-1.5** • *(1)*	**M-1719-1.5** • *(1)*
Holding torque	oz-in	32	60	75
	N-cm	23	42	53
Detent torque	oz-in	1.7	2.1	3.5
	N-cm	1.2	1.5	2.5
Rotor inertia	oz-in-sec^2	0.000538	0.0008037	0.0011562
	kg-cm^2	0.038	0.057	0.082
Weight	oz	7.4	8.1	12.7
	grams	210	230	360
Phase current	amps	1.5	1.5	1.5
Phase resistance	ohms	1.3	2.1	2.0
Phase inductance	mH	2.1	5.0	3.85

Figure 15: Nema 17 manufacturer specifications (Schneider Electronics)

3.2 Design of the electrical circuit for the control of the electric motors

With the selection of the electric motors most suitable for the application, the electrical circuit for their control should be designed. The fact that these motors are extensively used in commercial and prototyping applications, means that there are abundant resources on the control of these motors.

The movement of the stepper motors is possible via an intermediate circuit called a motor driver. Motor drivers for stepper motors, coordinate the current passing to each phase of the stepper motor, thus controlling the number of steps, and the direction from which the current

passes through each phase, thus controlling the direction of rotation. The information on the steps and the direction are sent by the microcontroller to the motor driver, which then executes the command by allocating the passing electric current accordingly.

There are various motor drivers suitable for the Nema 17 motor. More costly drivers such as the TMC2209 (Prodanov et. al., 2022)offer reduced noise of operation by decreasing vibration through advanced current handling, sensorless homing by monitoring the load of the motor as a function of the current drawn by it and many other functionalities that are not deemed necessary for this application or can be otherwise achieved with lower costs.

From the remaining options the most suitable motor driver is the DRV8825 circuit (Arief Wisnu Wardhana et. al., 2019). It offers 1/32nd microstepping capabilities to the motor, which translates to 0.06-degree theoretic accuracy and significant reduction on the noise during operation.

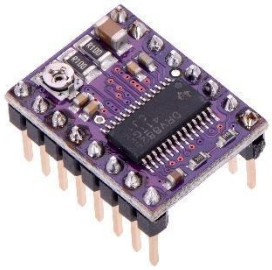


Figure 16: DRV8825 Motor driver breakout board

The motor driver is directly controlled by a microprocessor. The microcontroller passes a signal containing the number of steps and the direction of rotation for the stepper motors and the motor driver executes the motion. The only desired specification from the microcontroller in regards to the control of the stepper motors is the compatibility with the DRV8825 integrated circuit chosen, which is satisfied.

Further required specification from the microcontroller regarding the available memory and the communication protocol with the computer unit. Both of these features are thoroughly analyzed in the control software chapter; however, it is worth mentioning that a microprocessor with sufficient memory is required to store the control commands that respond to certain operations of the automated station and the communication with the computer unit should be via USB to avoid unnecessary serial communication equipment.

Both of these prerequisites are met with the Arduino Uno (Badamasi, 2014)microcontroller station. With 32 kbytes of flash memory it can store a program many times larger and complex than the one intended for the control software and the 2kbyte of RAM are more than enough for the calculations and the communication with the computing unit. Additionally, it is equipped with a USB-to-Serial converter, thus allowing direct serial communication with a computing unit via the USB port. These benefits, combined with its extensive use on prototyping and online support, make it an ideal microcontroller for this application.

Figure 17: Arduino Uno development board

The connection of four motors, two for the X-axis, and one for each of the Y and Z axis, to four DRV8825 integrated circuits and to the Arduino Uno could be made either on a common breadboard or on a custom PCB on a perfboard. The first approach works great for initial prototyping but it is neither safe nor easy to use. The second approach, while time consuming, is necessary for any permanent circuit to operate safely both in respect to the circuit itself and the user that operates the system. In any other case such a custom PCB would have been made for the purposes of the project. However, the extensive community of the Arduino microcontroller and the equally extensive use of stepper motors similar to the ones selected for the automated station, means that such PCBs are commercially available at a very reasonable cost. Additionally, they have been designed to attach directly on the Arduino UNO and have connectors available for the motor drivers and the cables of the stepper motors. Such integrated circuits that attach directly on the Arduino UNO, are called shields and the shield used for this purpose is called an Arduino GRBL V2 Shield (Hasan et. al., 2018).

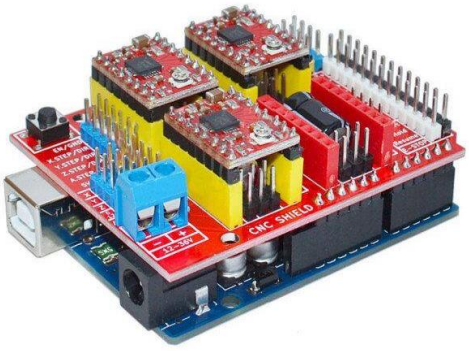


Figure 18: Arduino GRBL V2 Shield attached to an Arduino Uno

Lastly, a power supply unit is required to provide power to all individual components for the automated station. For the purposes of this thesis, it has been assumed that any computer unit (PC or single board computer) used for the control of the automated station will have an independent power supply.

The power supply unit suitable for this application is selected based on three parameters. Firstly, the DC voltage it outputs, the maximum amperage it can deliver and the combination of these two parameters, the maximum power output in Watts. In regard to the DC voltage, the power supply is usually selected to match the voltage of the component that has the greatest power demand. Components requiring lower voltage can be accommodated via the

use of an intermediate voltage converter, also known as buck converter, which is an integrated circuit that lowers the voltage provided by a power supply. A voltage converter can also be used to power components requiring higher voltage, though they dissipate significant heat, even in low to moderate amperages and are best avoided and the increased voltage requirement is met by a secondary power supply unit.

In the scope of this application, the most power intensive components are the stepper motors, which are running on 12 volts. The Arduino UNO can also operate in that voltage, as well as the motor drivers and GRBL shield, so there is no need for voltage conversion.

Given that the voltage is the same for each of the components presented above, the demand in amperage and by extension in power, can be easily calculated. For the amperage the maximum amperage stated by the manufacturer of each component is added. The combined amperage is then increased by a safety factor of 10%, and this is the required maximum amperage the power supply can deliver. The required power output of the power supply is the result of the multiplication of the maximum required amperage and the DC voltage output of the power supply.

$$\max(A) = (A_{Arduino} + 4 * A_{Nema17}) * 1.1 \quad (16)$$
$$\max(A) = (0.04 + 4 * 1.8) * 1.1 = 7.964\text{A}$$

With these specifications in mind, a power supply unit is chosen from any of the commercially available ones, such as the one illustrated in figure 19.

Figure 19: Power supply unit 12V, 8A

3.3 Additional electrical parts and wiring

Within the design of the electrical circuit of the automated station, there are certain secondary components that are important for the proper function of the overall assembly.

Firstly, as mentioned previously, the selected stepper motors operate in an open loop. That implies that the controller can position the motors, but there is no feedback on whether the position has been reached. This could be potentially problematic if a moving parts motion is obstructed for any reason, when the motors will continue to operate until the duration of the movement is seemingly complete, even though there is no actual movement taking place. Given the nature of the application, it is highly unlikely that an obstacle will be presented to the motion of any of the axes, as they are designed with the volume of the object being inspected in mind. Thus, no need for a closed loop system, one with constant position feedback is deemed necessary.

However, the same assumption cannot be made in the case of an abrupt loss of power.

Through the control software, it is possible to save the position of each axis during the closing sequence of the control software and retrieve that position during the opening sequence. In case of unexpected loss of power, the software does not have the opportunity to save that position and during the startup sequence it will assume that the axes are either at the home position or the last known closing position. The same could also occur if the axis are moved even slightly while the automated station is unpowered. The result of something like that would be the controller not being aware of the actual position of each axis, which would cause catastrophic failure of one or more parts of the assembly in case they reach the end of their corresponding lead screw in high velocity.

To avoid any of the above to occur, it is deemed necessary to have a homing sequence, during which the axes are moved to their zero position and their position, as information, is updated on the controller. That is possible with the installation of endstop switches at the ends of each lead screw. This not only ensures the safe initialization of the automated station, but also solves two other security risks. The first one is the case of obstacles in the way of an axis, as illustrated above, where the motor unable to proceed skips steps. In that case, even if an axis of the automated station, having skipped steps, reached towards the end of the lead screw, even at significant velocities, a properly placed endstop can prevent it from ramming onto the structural frame. Additionally, in case the X-axis lead screws and their corresponding nuts are misaligned for any of the reasons already mentioned, the endstops can provide a quick way to fix such a problem.

These endstops are nothing more than a switch that when pressed sends a signal to the controller. There are endstops switches commercially available that provide additional safety functionalities but for the purposes of this application, a simple endstop switch, similar to the one illustrated below, installed at the two ends of each lead screw is sufficient.

Figure 20: Endstop switch

Equally important secondary electrical components are the connectors and the wiring that connect the individual components of the electrical assembly. In regard to the wiring, depending on the amperage passing through it, specific diameters, also known as gauges, of wire are required. For the purposes of this application, there are two different kinds of wiring. One is the signal wiring, which as the name implies, transfers signals between the individual components of the electrical assembly. Usually, they operate under 5V and the current passing through them is of the mA scale. Such wires are of the smallest diameter, yet shielded enough not to cause interference from the current passing from nearby cables. The other type of wiring is the power supply wiring. As mentioned, power supply voltage for this application is 12V and the current passing through these wires is of the A scale. For each of these two types of cable, a corresponding diameter is chosen according to safety standards (Locke, 2008).

In regard to the connectors, their function is to ensure the secure connection of cables to the appropriate electrical components in a way that is safe yet does not restrict any possible

changes to the electrical assembly during prototyping. In any other application, where there was no prototyping element, the connection would be permanently secured via soldering. The closest alternative was the use of connectors, and specifically JST connectors (Advanced Connection Systems - JST catalogue Vol. 120e)that are suitable both for signaling cables. A typical JST connector is illustrated below. For the power delivery to the stepper motors, the JST connectors are only borderline suitable, given the amperage range of each motor, so for safety reasons the more suitable 4-pin GX connectors (Hanson Technology - GX Aviation Connector Datasheet) were instead chosen and are also presented in figure 21.

Figure 21: JST and GX4 connectors

Lastly, in regard to the wiring it should be noted that the color scheme for the cabling was taken into account. For the power delivery cable for each phase of the stepper motors, the traditional color scheme for their cabling was upheld, meaning the combination of black, red, blue and green wires. For the signaling wires, the color yellow was chosen as it is most commonly used in data transferring cables. Additionally, since the major electrical components are fixed in certain positions on the structural frame, the wired connecting them could easily interfere with the free movement of each axis, if left unattended, thus posing a serious safety danger. It was deemed necessary that a cable harness be installed along each moving sub-assembly, to allow for the unrestricted moving of the axis, while safely storing all relative wiring.

3.4 Assembly of the electrical circuit

With all the major and secondary components selected, it is possible to design the final layout of the electrical circuit assembly. The assembly is firstly designed virtually to ensure, one last time, the compatibility of the individual components. The most appropriate software is Fritzing (Knörig et. al., 2009). The end result of the assembly on this software is illustrated in figure 22.

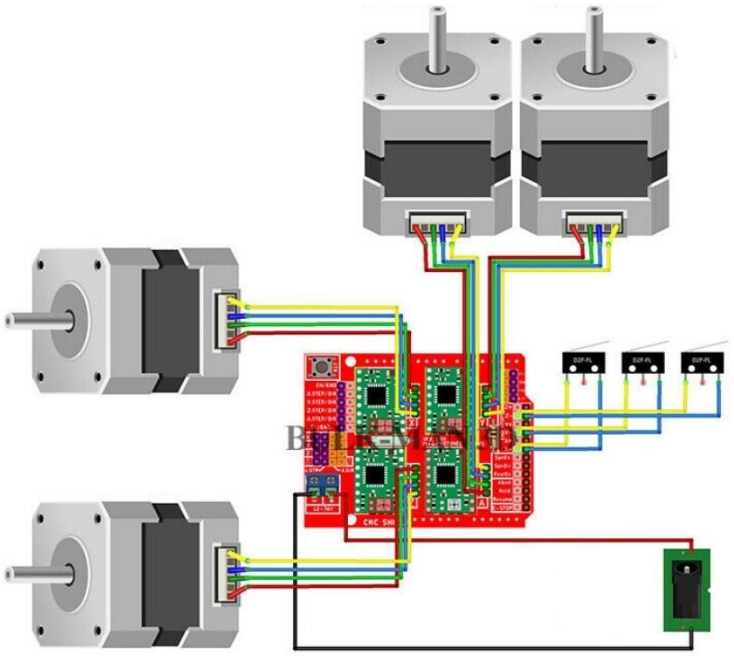


Figure 22: Fritzing electronics diagram

After purchasing the components, taking into account the physical dimensions of the mechanical assembly for the proper lengths of the wiring and the cable harness, it is possible to proceed to the physical assembly of the electrical circuit. It should be noted that the Arduino Uno along with the GRBL shield, the motor drivers and the GX connectors for the motors are enclosed in a 3D printed case both to be securely attached to the structural frame but also to ensure the safety of the operator during use. For the same reason, the power supply unit is also housed in a 3D printed enclosure. Both of the enclosures were designed in a CAD software, the files were exported in .STL file format ad were printed using an FDM printer with PLA filament. The technical drawings of these parts can be found in Annex A.

Figure 23: 3D printed enclosures for Arduino and PSU

With the physical assembly of the electrical circuit on the already assembled mechanical part, the automated station is presented in the photographs of figure 24.

Figure 24: Final assembly of automated station - Photographs

4. Control software for the automated station

The process of writing a control software for the automated station, given the complexity of the task at hand, is not possible without an initial conceptual model of the structure of the individual parts that comprise the system. This model, commonly referred to as flowchart will serve as a much-needed guide during development, due to the complexity of the task, and as a means of explanation of the algorithmic process. The flowchart mentions processes and details relevant to the developments of the software that will be thoroughly explained on the chapters that follow.

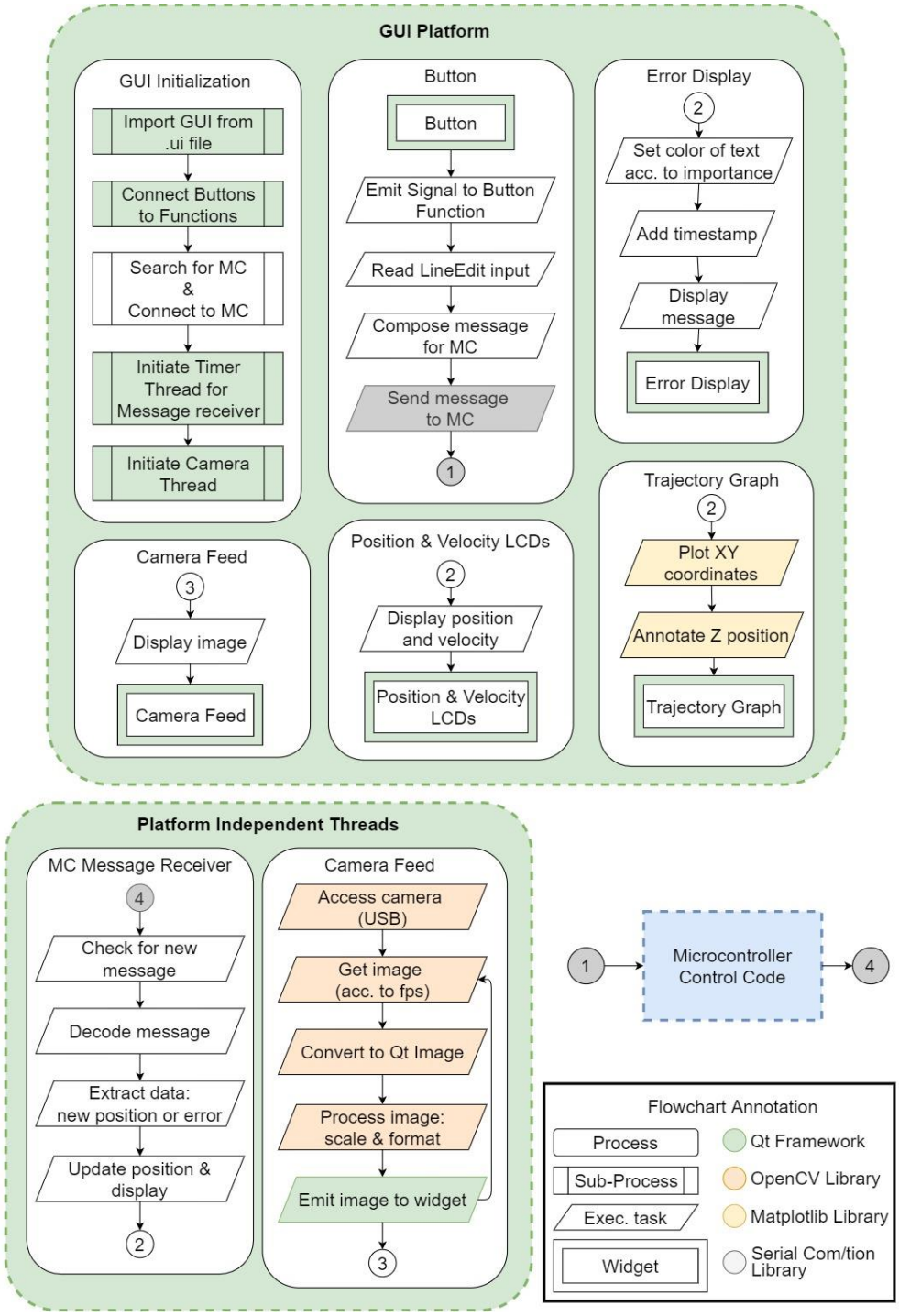


Figure 25: Control software flowchart

The specific flowchart illustrated in Figure 24, is a graphical depiction of the processes of the control software designed for this application. Firstly, there are two distinct sub-divisions of what has been thus far referred to as control software, meaning the one developed for the PC and one for the microcontroller. The former functions as a way for the user to interact with the control software via inputs and outputs on the computer screen, which is the main function of the GUI. It also includes all other sub-processes that enable the user to send command to the microcontroller and by extension control the automated station. The latter, as the name implies regards to the processes allowing the microcontroller to directly control the individual electronic parts of the automate station or receive input form them. The purpose, structure and implementation of these processes will be thoroughly explained in the following chapters.

4.1 Graphic User Interface

For the creation of the Graphic User Interface (GUI), it is best to choose a tool with a wide variety of applications and widespread use among developers. Both of these characteristics will allow for the integration and possible adaptation of the different modules to the GUI. The industry standard for the creation of complex, scalable GUIs is, arguably, the Qt framework (Mezei, 2017).

The Qt framework allows for the development of cross-station applications that run on all major desktop stations and supports various compilers. Though most notably used with C++ compilers, lately there has been widespread adoption of the pyQt library (Willman, 2022) which adapts the Qt framework for use in Python applications. The use of C++ for the Qt framework, would most likely result in an application that runs much faster than an equivalent application running pyQt on Python. However, the various functions of this application that run within the GUI, such as the communication with the microcontroller, are not easily implemented in C++ by one not adept to the language, as C++ has a notoriously steep learning curve. For these reasons, the implementation of the pyQt framework on a Python application was deemed as the most suitable and most specifically its more widespread and stable version, pyQt5.

In regards to the Qt framework, it is important to illustrate its structure and how the various components of the control software will be developed along this structure. In the Qt framework, a window is created by the combination of different widgets. Widgets are elements that can display data, receive user input or simply provide a container for other widgets to be grouped in. Depending on the needs of the application, the designer/developer chooses the most appropriate widgets to integrate in the main window and groups these widgets within other widgets to allow for an adaptable GUI, in regard to its size. These widgets, placed in specific positions, can be customized to allow for a more user-friendly environment.

The main window can be structured as a main function within the UI python file of the python application. Though such an approach is direct and offers a lot of control to the developer, for more complex applications, not unsimilar to this one, the sheer volume of the commands for the creation of the main window interface quickly becomes staggering. For this reason, the creators of the Qt Framework have developed a software called Qt Designer. Within Qt Designer, widgets can be placed on the main window via drag-and-drop and further customized within the environment of the software. Once the placement and customization of the widgets is complete, the software exports a file (.ui) which can be

converted to executable Python code. This process allows for a much more intuitive design of the front end of the application being developed, while significantly reducing the complexity and time requirement.

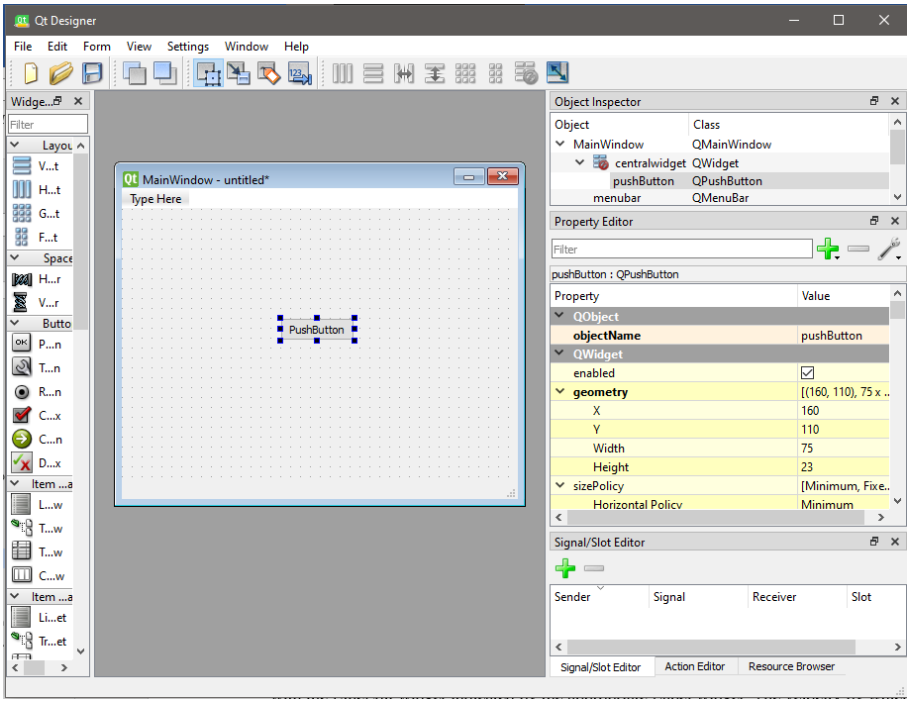


Figure 26: Qt designer environment

With the use of the Qt designer, the main window of the control software is developed. The first step is the selection of appropriate widgets. For the control of the movement of the end tool (camera), 4 buttons are selected to correspond to the X and Y movement and are positioned in a cross to mimic the movement of the camera within the horizontal plane of the working area. For the Z movement, two buttons are positioned corresponding to the upwards and downwards motion of the camera along the vertical axis.

The increment by which each axis is moved when a button is pressed is inputted by the user with the LineEdit widget indicated by the appropriate Label widget. The velocity by which the motion is carried out is also indicated by a Label widget and inputted with the selection of a RadioButton corresponding to slow movement and another one corresponding to fast. The options regarding the velocity are kept that way to limit the available options of speed to the limit as without proper knowledge of the capabilities of the system, an incorrect input could be potentially catastrophic for a subsystem or the automated station as a whole. The widgets mentioned above are grouped and labeled as Manual Control.

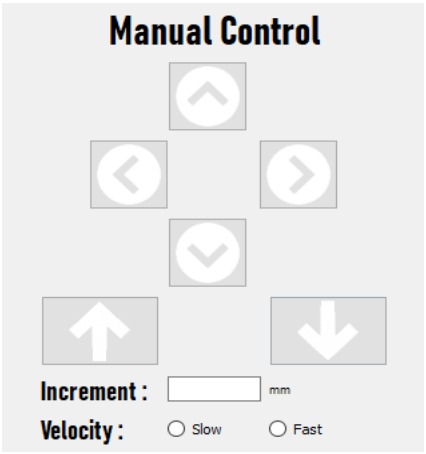


Figure 27: Manual Control Group

For a more precise movement, a group of widgets is created and labeled as Input Control. This group includes Label widgets indicating each axis and underneath each one, a LineEdit widget is placed, intended for the input of a numerical value by the user. Next to these there are two RadioBox widgets indicating if the movement the user wishes to initiate will be in regards to the relative or the absolute coordinate system of each axis, meaning whether the end tool will move by a certain distance in each axis or to specific coordinated, according to the numerical value inputted in the corresponding LineEdit widget of each axis.

Below the Input Control group, another group of widgets is placed, which includes the buttons for the main automated functionalities of the control software, meaning the ones not included in the Manual Control group. These functions are:

- Homing function indicated by Label widget as “Home”
- Stopping function which ceases any ongoing movement and is labeled as “Stop”,
- Photographing function, which saves the image that the camera is currently capturing as an appropriate file and is labeled as “Photo”
- Moving function, which initiates a movement according to the specifications of position and velocity inputted by the user in the Input Control widget group and is labeled as “Move”.
- “Save” and “Load” buttons, which for the purposes of this thesis are programmed to save the last known coordinates of the end tool and load them in the software upon initiating it.

The buttons for each function have an image embedded that represents the purpose of the function, thus making the use of the control software much more intuitive.

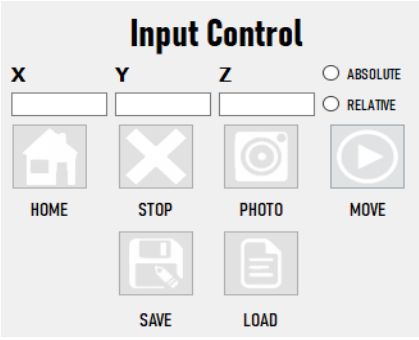


Figure 28: Input Control and Main buttons

Additionally, label widgets are positioned to indicate the current position of the camera, corresponding to the X, Y and Z coordinates and each intended velocity. The numerical values for this data are outputted to the user in LCDNumber widgets, to differentiate the visually from the data inputted by the user.

Figure 29: LCD displays for XYZ

A Tab widget is used to group the two main outputs of the control software, the position graph and the camera feed. The Tab widget is selected because it allows the user to switch between these two elements, which both are large enough that if placed simultaneously on the main window they would be overwhelming to the user, require a large screen size and overall diminish the ease of use of the GUI. In place of the camera feed and the position graph, during the design of the GUI, a QWidget is placed, which is an element that reserves this position in the main window open for the developer to add custom content.

Figure 30: Tab widget

Lastly, on the bottom of the main window a QPlainTextEdit widget is placed. This widget is practically an open editable text window. Its function within this application, however, is to display all the necessary feedback to the user that cannot be otherwise displayed by the other widgets. This information could be the successful completion of a function, such as movement, photograph etc. or the occurrence of an error that needs to be addressed by the user. Its editability is therefore blocked for the user and the text it displays is provided directly by the control software.

With all the necessary components of the GUI placed and grouped, there is one last step needed to ensure an optimized main window. That is the placement of spacers, horizontal and vertical ones. These spacers serve an important role, which is to keep the spacing between grouped widgets set to certain values, regardless of the size of the main window. Without these elements, the placement of each widget would differ according to the size of the main window, thus resulting in a very confusing and difficult to work layout. With these

spacers added, the final GUI, as exported by Qt Designer, is illustrated in figure 31.

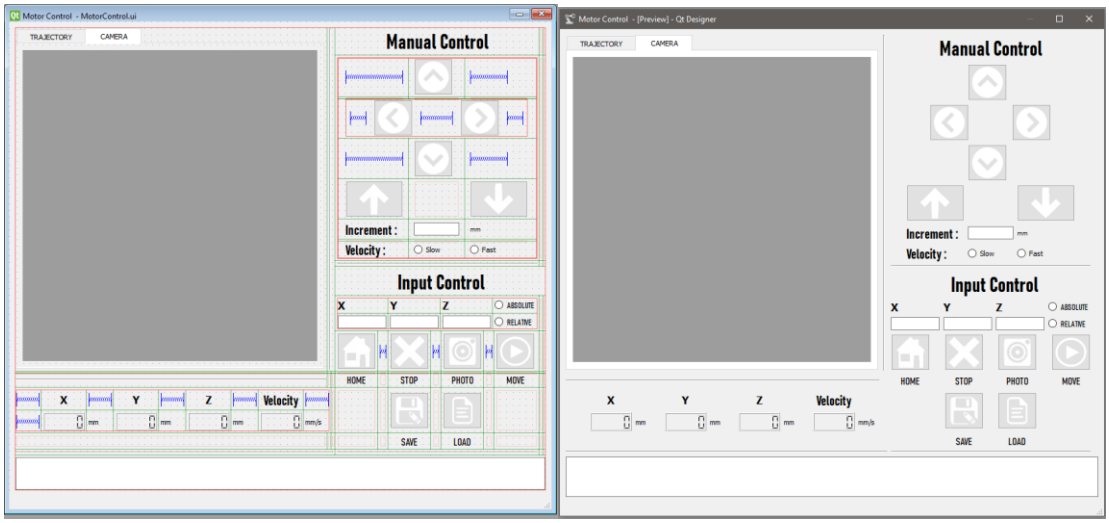


Figure 31: GUI as seen in Qt Designer (left) and as exported from Qt Designer (right)

The file type exported by Qt Designer is not, at this point, fit to be run within a Python script. For that to be possible a conversion of the file is required, to a Python script. That can be accomplished with the pyuic5 development tool, which converts the .UI (Culjak et. al., 2012) file to a python script with a pyQt5 syntax. After this conversion, the resulting file includes, in order, the necessary imports, meaning the libraries needed for the script to run, and a main function which includes all the widgets that were selected in the designing phase. Each widget is listed with its appropriate placement and any customization made within Qt Designer, as an editable python code in pyQt5 syntax. This code can successfully run, however since no functions are attached to the widgets, no results will be outputted.

At this point any further customization on the main window or individual widgets that was not able to be made within the environment of Qt Designer can be made. Since there is no need for any customization, within the purposes of this application, the only change deemed necessary is the overall color scheme of the main window. At its current state it is based on light theme colors, which could be problematic after long term use of the software due to eye strain issues from high brightness. For this reason, a dark theme is applied to the overall color scheme of the application. The final GUI, after the placement of all individual widgets, the conversion to Python script and the customization, can be shown by running the relative Python script and its output is illustrated in figure 32.

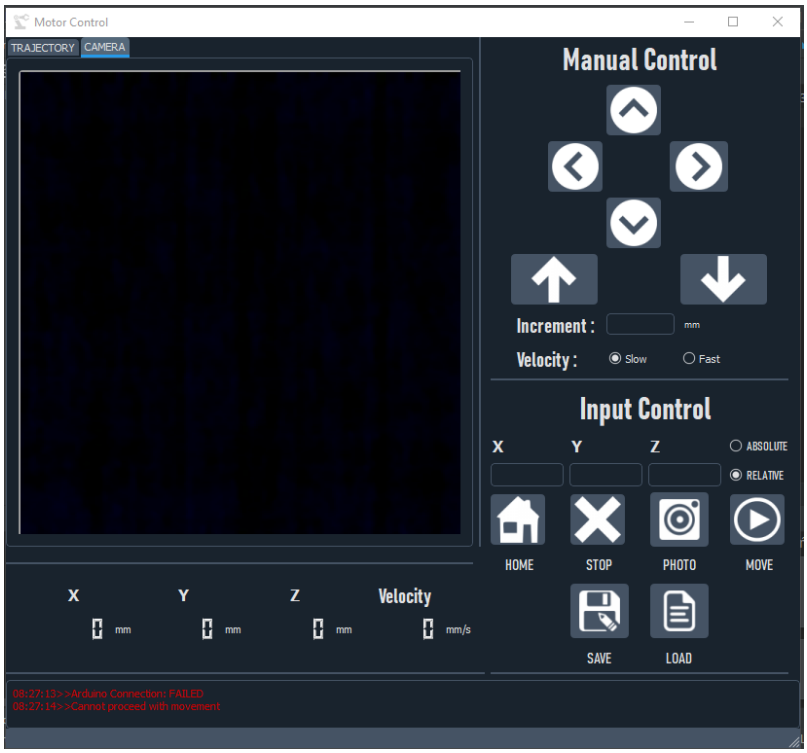


Figure 32: Final GUI from Python script

4.2 Motion control of the automated station

As mentioned in the previous chapter, the GUI works as an interface between the user and the control software, with which inputs and commands are given to the software and certain processes are executed by the software and the subsystems it is connected with, with the results of these processes being outputted as feedback to the user via the GUI. The framework within which the GUI is designed operates in a very specific mechanism that makes that possible. This mechanism is referred to as Signals & Slots.

Within the Qt framework, every widget has a set of pre-defined signals it can emit, when a particular event takes place. A slot is a function that is called in response to a particular signal. Slots are functions, most commonly defined by the developer, that perform a specific task. In simple terms, while the Python script is running the main function that displays the GUI is constantly checking for events. Once an event occurs, it emits the signal that is linked to that event, that runs the function appointed to a specific Slot. Such an event could be the pressing of a button, which would signal a function to run that performs a task appointed on the pressing of the button.

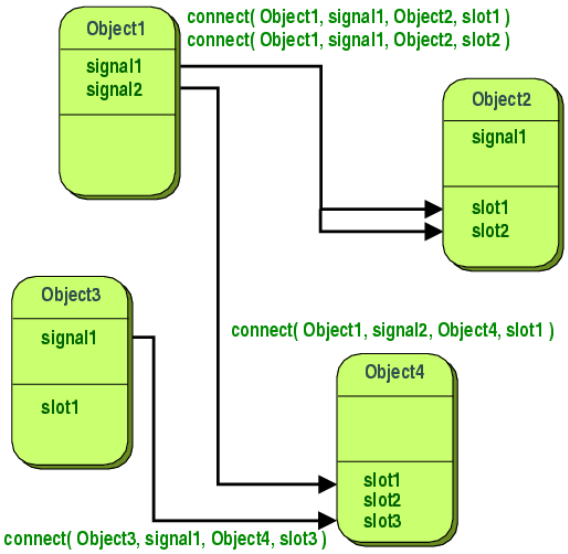


Figure 33: Slot and Signal graph of Qt framework

With this in mind it is relatively straightforward to attach a specific function to each button. Initially, the press of a button is connected to a function that, depending on the button, reads the data inputted by the user on the purpose of the button and sends an appropriate command to the microcontroller to be executed. This seemingly straightforward process has a few obstacles that need to be addressed, depending on the functionality of each button. First and foremost, the communication with the microcontroller.

As mentioned in a previous chapter, a computer is not capable of creating the output signal necessary for the control of the motor drivers and by extension the motors or to receive the input from the endstop directly. As such, there is a necessity for a microcontroller that intermediates between certain electronic components and the computer and translates the commands given by the user, through the GUI, to electric signals that result in appropriate actions. The communication between the microcontroller and the computer is therefore an integral part of the control process and should be addressed within the control software.

The most common communication between a computer and an external device is via USB connection. The microcontroller chosen for this application has a USB-to-Serial converter which allows for serial communication between the computer and the microcontroller. To enable this communication the appropriate library must be added to the imports of the scripts. The Qt framework offers the capability of serial communication, further reducing the need for any additional library.

In regard to the microcontroller, which by default is programmed with the C++ language, after providing power to it, a setup function runs once, which amongst other things, initiates serial communication with the computer at 115200 baud rate. After the setup function runs once, another function runs on a loop. Inside this loop function, there is a command that checks whether a message is received through the serial port. What this function does with the message received, will be fully explained further below.

At this point, any moment a button is pressed, the computer can send a specific message to the microcontroller and the microcontroller constantly checks if a new message has arrived. The opposite has not been programmed, meaning the microcontroller cannot pass

information to the computer. That is problematic, as there is no confirmation on the execution of commands by the microcontroller. The reason the opposite communication is much harder has to do with the very structure of the main window within the Qt framework.

As previously explained, the main window is a block of code constantly running to check for events that can trigger certain signals. As such, the programming of a constant check for received messages is difficult, as an open WHILE loop would cause the program to not respond to other incoming events. To overcome this, after consideration of the available tools provided by the QT framework, the most suitable solution was determined to be the use of a QTimer. This is a class which, as the name implies, creates a timer and is connected to a slot. The developer inputs a specific time interval, which is the numerical value in milliseconds the timer is counting down from. Upon reaching the end of the countdown (timeout), a signal is emitted, which activates the slot. The advantage is that the timer runs in parallel with the main window and therefore does not affect its operation. With this process, an interval of 300ms is chosen, as no command of the automated station is expected to be executed in less time than that, and the communication from the microcontroller is checked every time timeout is reached.

The last matter to be addressed in the communication between the computer and the microcontroller is the message itself. As already stated, the commands the microcontroller executes are not fast enough to interfere with the communication with the computer. Since these commands are mostly in regard to physical movements of the automated station and the baud rate is high enough, there are near zero chances a message from the microcontroller to the computer will be interrupted by another message, resulting in a fragmented final message received by the computer. The same cannot be said for messages sent by the computer to the microcontroller. With fast enough clicking of the buttons, messages can be sent at a speed that can cause issues with the integrity of the commands sent. Though in such cases the command will be ignored by the microcontroller, as it will not fit within the messages it is expecting to receive, such occurrences must be avoided. The simplest way is to use the smallest possible message. Given the fact that baud rate is essentially the number of symbols that can be transmitted in a second, then the smaller the message, the less time required to be fully transmitting, thus minimizing the risk of fragmentation.

To create the smallest message possible, only the most necessary information regarding a specific command must be included in it. For commands regarding movement of the automated station the smallest message must include the button pressed, the x, y and z values, the velocity and whether the movement is in absolute or relative coordinates. The values are separated by a denominator, in this case a comma, to make parsing the data easier. For the homing sequence and the stop command, only a single numerical value appointed to each process is enough to initiate each function.

Having established successful communication between the computer and the microcontroller, each button can be programmed to accomplish a specific function. The programming of each button is rather similar. The button, when pressed, reads from a specific LineEdit widget a user input and creates an appropriate message, which the passes to the function that communicates with the microcontroller. This applies for the manual control buttons, the Move, Stop and Homing Buttons.

In regard to the Photo button, its programming requires the explanation of the operation of the camera feed, which is the subject of a later chapter. The Save and Load buttons, as explained, respectively save the location of the last known X, Y and Z coordinates and load

them to the software by the user, both fairly simple programming tasks.

4.3 Programming the microcontroller

Programming the microcontroller that executes the motion commands provided by the computer, is a much different task than programming the GUI and its functions. As already stated, the very structure of the two programs is vastly different. A programming script for a microcontroller similar to the one used in this application usually consists of three different sections.

The first section includes the inclusion of all necessary libraries for each application, the declaration of the type and initial value of variables and constants and of any other programming element as required by a library. The only noteworthy steps in this section are the selection of names for each of the aforementioned elements and their proper documentation with comments, as both of these steps allow for a much cleaner code.

The second section is the setup function. This function only runs once, upon providing power to the microcontroller. It includes the initialization of pins and their appropriate roles as inputs or outputs, the initialization of the serial communication, and the setup of the maximum speed and acceleration for the stepper motors.

The third section is the loop function, which, as the name implies, runs in a loop. Within the loop function are all the commands the microcontroller executes in repeat. This is the part that mostly differentiates the programming of microcontrollers to the programming of an application like the GUI with Python and Qt. Underneath the loop function, there can be various other functions created by the developer to execute specific tasks, but only run if they are called by the loop function.

Within the loop function, there is a command that constantly checks for new messages that might have been received from the computer. Should a new message exist in the serial communication, it is sent to a function that converts it into a string type. This string is then passed to a parsing function that breaks the string according to the denominator chosen and saves them in specific places inside an array. A button function is called that, based on the element of the array that regards a specific button and the elements of the array that regard the information of the movement that needs to be executed, updates the new position of each axis. The new position of each axis is then passed on a function that executes the movement, meaning it commands the motors to be moved to the updated position. In case the button function identifies in the appropriate element of the array the value that indicates the stopping or homing function it acts accordingly, following the same process described above.

Once the command has been executed and the process (movement, stopping or homing) has been completed, a messaging function sends a message to the computer to inform the successful completion of the movement. A timeout condition exists on the control software of the computer that indicates that if a command has been given and no response has been returned within a specific time interval then the movement has been unsuccessful and an error is displayed in the GUI.

By using simple coding practices, such as the switch statements, the overall code can be both easily read but also minimal. This minimization of redundant complexity in the coding of

the functions mentioned above, resulted in use of less than 30% of the microcontroller's storage space and 25% use of the dynamic memory, leaving ample memory for computation and data handling.

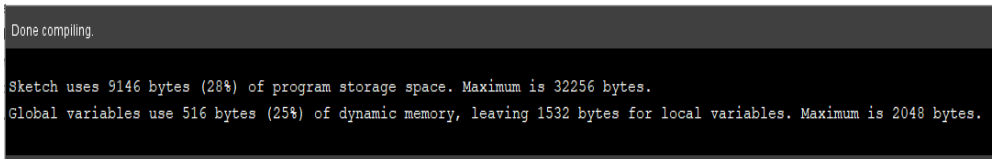


Figure 34: Compiled microcontroller code size on the Arduino IDE

4.4 Position graph, error dialog and camera feed

Programming the position graph is not dissimilar to the programming of the button. As mentioned, the position graph will be implemented within the relative QWidget. A QWidget is a subclass within the Qt framework that allows for the custom creatin of widgets, meaning ones not already provided by the framework. Within this QWidget the position graph is programmed using the matplotlib, a library used extensively by developers for graphs of any kind.

The graph is implemented in the most intuitive way, meaning the vertical axis of the graph represents the Y axis of motion and the horizontal axis represents the X axis. This is similar to how the user sees the actual axis in motion when observing the automated station from above with its longer side, the X axis perpendicular to the user. The extends of each axis of the graph are numbered according to the physical extends of the relative axis they represent. The Z axis cannot be represented directly, since that would require a 3D graph, which is arguably much less intuitive. The Z position of the camera is therefore displayed as a numerical value next to the point indicating the current position of the end tool – camera on the position graph. The zero point has already been selected during the development of the homing code, and is set accordingly on the position graph. The position graph at a random point, during operation of the automated station, is illustrated in figure 35.

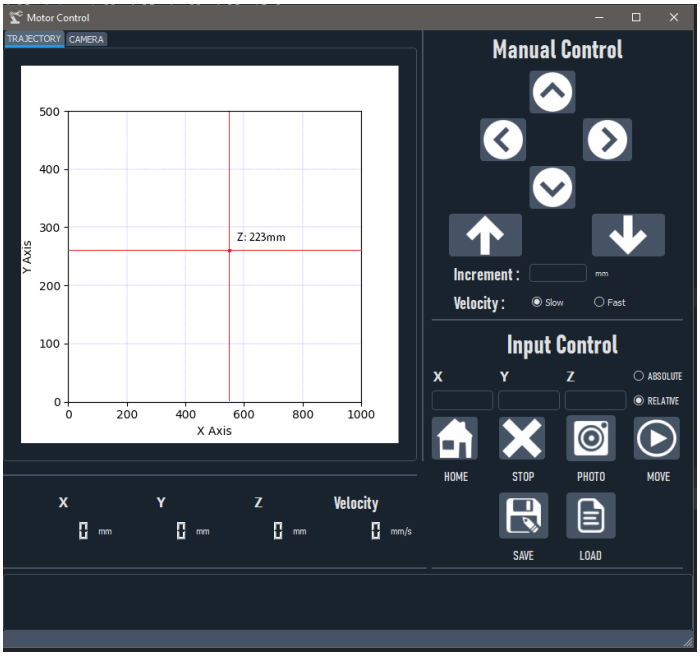


Figure 35: Position graph example

To avoid unnecessary complication and since there is no need for constant update of the position of the end tool, the X, Y and Z positions are only updated when the control software of the computer receives a message from the microcontroller containing information about the completion of a movement. Therefore, the values displayed in the graph are always true to the microcontroller and are only updated when there is need to do so. Within the Qt framework that means that the function receiving messages from the microcontroller, as was explained in an earlier chapter, is added the task of checking the content of the message and should it include the successful completion of the movement, which in Qt framework terms would be the event, a signal is emitted to the position graph function, which is the slot, and it updates the graph accordingly.

Since the function receiving messages has been updated to inspect the content of the messages received, it can now be tasked with the update of the error display. As mentioned previously, a QPlainText widget has been placed on the bottom of the main window to inform the user on the completion of each task. A successful completion of a task is more or less displayed by the rest of the widgets, whether it is the update of the position or the velocity. However, this information can be potentially missed by the user, for example in the case of a small enough movement, and additionally there is no way to examine it retrospectively. An unsuccessful completion of a task cannot be displayed with any other widget. So, there is great importance for direct communication of the processes of the microcontroller and their status. This function is carried out by the QPlainText widget placed at the bottom of the main window, which will henceforth be referred to as error display.

Every message transmitted from the microcontroller the receiving function decodes it, and based on its content, it is sent to the error display function and is displayed using a matching color; successful tasks as displayed in green, information critical to the user, but otherwise not problematic to the function of the automated station are displayed in yellow and critical errors are displayed in red. Next to each message displayed in the error display, there is a time stamp indicative to the moment the task or error was displayed for logging purposes. In Qt terms, the information of the message decoded by the receiving function is the event, which triggers a signal that activates the slot, which is the error displaying function that updates the error display accordingly. During operation, the error display might contain the information presented in figure 36.

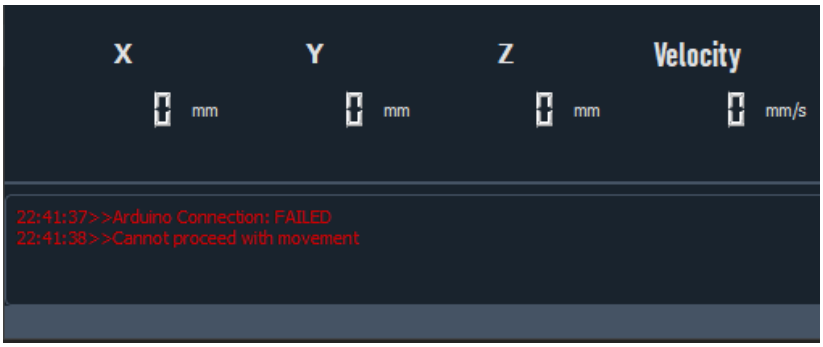


Figure 36: Error display during operation of the automated station

Lastly, the only part of the control software that remains unmentioned is the camera feed. Given the importance of the camera feed in this application, the camera feed is arguably the most important, and though it is the last presented in order, the rest of the control software has been written in a way that facilitates the proper function of the camera feed.

As already mentioned, the camera feed will be placed within a QWidget that is embedded in the Tab widget of the main window. Within the Qt framework, there are capabilities that allow for a camera feed to be implemented within a widget, however such implementation is very limited in the scope of this application, where image processing might be required depending on the object inspected by the automated station.

A camera feed is simply a timed collection of a single image at specific intervals (frames per second). This collection is possible through specific communication to the camera. The camera chosen for this application has a USB connection. A communication process similar to the one implemented for the microcontroller could be theoretically possible. In such a case, a timer could be set and the request for an image by the control software could be made at specific intervals. The image collected could be then displayed to the QWidget, by properly customizing its functionality, and the process could be repeated to achieve live feed for the camera.

While this could be a potential implementation, it is rather unrefined. A much less complex and common implementation would be with a class of the Qt framework called QThread. This class, as the name implies, is an object that manages one thread of control within the program, basically resolving the inherent difficulty of running looping actions within a Qt created GUI. In terms of the Qt framework, an event loop can run within the thread concurrently with the main window, once activated.

For the collection of the images, the most widely used library is OpenCV (Culjak et. al., 2012), an open-source computer vision library. By combining all the above, the camera feed is programmed by customizing the QWidget to display an image received by the QThread. The QThread is activated within the QWidget and is programmed with the OpenCV library to receive images by a camera device connected in a specific USB port and with a specific color scheme, aspect ratio etc. Every time a new image is collected, which is regarded as an event, a signal is sent to the QWidget, which displays the image. This process is fast enough to result in live camera feed.

With the use of OpenCV, another function of the control software is rendered possible. That is the collection of images. Within the camera QThread, there is a global flag that once activated saves the current image displayed by the camera with an appropriate name. The flag is activated with the press of the “Photo” button. In the Qt framework, the event is the press of the button and the signal is sent to the thread (instead of another function of the main window as any other button), which acts as a slot. Lastly, in the camera feed, utilizing the OpenCV capabilities, a crosshair is placed in each frame, meaning two intersecting lines forming a cross to enable the user to easily place the camera in a specific position.

With the completion of the programming of all the major functionalities of the control software, several other elements are programmed. These elements, however, regard secondary or assisting tasks and do not require any sophisticated implementation. Examples include a pop-up window during closing to inform the user of termination of the GUI, the function that encodes the message sent to the microcontroller and others. With the completion of the programming of these functions as well, the control software can be considered complete, and the collection of images for defect detection is now possible.

5. Implementation of defect detection using Artificial Neural Networks

This chapter aims to illustrate the process of utilization of latest advances in the field of A.I., more specifically of ANNs for defect detection on certain aluminum cast items. The immense complexity of the field is the first hurdle towards its proper utilization. It is, however, arguably the most important step, as attempting to implement Artificial Neural Network tools in an application without insight on its functionality, will make any attempt to interpret and optimize its output potentially invalid.

Core concepts of ANNs will firstly be introduced and during the illustration of the implementation process, these concepts will be thoroughly explained within the scopes of the application. Any attempt to further delve into the details of the sub-processes of the operation and implementation of Artificial Neural Networks, will be avoided to keep the analysis short and on point.

Firstly, for defect detection, the most utilized field of A.I. is Deep Learning (LeCun et. al., 2015). Deep Learning is a subset of ML, which in it of itself belongs to the scope of A.I. ML aims to enable computers to complete certain tasks without explicit programming and Deep Learning aims to achieve this by utilization of ANNs. This difference between the two is actually immense in its core concept, as the utilization of ANNs allows the data to pass through the nodes of the network in highly connected ways, resulting in non-linear transformation of the data with increasing abstraction.

This abstraction is the very edge of Deep learning. Because of it, compared to other ML methodologies, Deep Learning has some inherent advantages. It can accommodate large volumes of unstructured data, produce output fast and accurately and most importantly be utilized in cases where increased complexity of the system, such as non-linearities or disturbances, would have to be otherwise dealt with traditional algorithms of slower speed and decreased accuracy, such as linear regression. Taking into account the data volume from a video or live camera feed and the demand for near real-time detection, the exponential progress and implementation of Deep Learning in object detection in the past ten years begs no question. The complexity and time demand of the set up and the increased demands in higher hardware specifications are the obvious disadvantages in the utilization of Deep Learning, but within the scope of this application they are deemed as non-problematic and are therefore disregarded.

Within the field of Deep Learning, the design of the ANNs, is referred to as architecture. The differentiation between the different types of architectures is a highly technical matter and the progress of the field is such that, to simply mention the most prevalent ANN architectures and their differences is well out of the scope of the current thesis. Instead, the focus will remain only on the architectures currently overutilized on object detection and disregard ones that are either still on a research level, not commonly used or otherwise not heavily favored by professional developers of the related field.

Before further elaboration on the ANN architectures for object detection, a proper definition of the term is needed. Within the field of A.I., object detection is a computer vision task falling under the general term of object recognition. Object recognition (Zhong-Qiu Zhao et. al., 2019) includes image classification (Al-Saffar et. al., 2017; Atraszkiewicz et. al., 2020) and object localization (Long et. al., 2017). The former is a process including the input of an

image with a single object and the output of a class label relative to the content of the image. The latter is a process including the input of an image with multiple objects with an output of a bounding box around one or more objects. Object detection is the combination of these two tasks within one process, which ultimately includes the object localization of one or more objects within an image with the addition of a class label on each object. With this definition in mind, it is a rather simple transition to defect detection. In regarding each type of defect as a different class of object and labeling it as such, with the utilization of ANNs, we can determine the location and type of a defect on a product.

In the following chapters, the different architectures of ANNs utilized in object detection will be presented, the most suitable will be selected and the implementation steps will be explained. Ultimately, the appropriate ANN architecture will be implemented and the resulting model will be integrated in an application for use with the automated station. The steps for the implementation of the model, if followed correctly and with respect to the different features of each application, can allow the automated station to operate as a quality inspection tool for a number of different applications.

5.1 Artificial Neural Network architecture

As mentioned, to delve into the characteristics of all different ANN architectures, and their individual distinctions, is far beyond the scope of this thesis, as the immense progress of the field the past decade demands the thorough explanation of highly technical terms. A simpler, more direct way to determine the optimal type of ANN to implement in this specific application, is simply to compare the most commonly utilized, in applications similar to our own, architectures and types and find the one most fitting.

Another, arguably more important, reason to look into the most commonly used ANN types is also a result of the highly technical nature of the matter and needs to be addressed. The implementation of a neural network, from its conception to the actual software development requires extensive knowledge on the subject matter and a high degree of skill in software development. While there are plenty of frameworks that simplify to an extent the development of a neural network model, the demand for the developer to be familiar with the extensive background regarding neural networks remains. The more a type of ANN has been utilized in real life applications, the more information is available for its proper implementation. The widespread use of a certain type of ANN in a certain field, can also rapidly decrease the time demands for the setup of an ANN, due to the existence of pre-trained models. These pre-trained models and their importance to object detection application will be thoroughly explained later on.

For object detection purposes, the most overutilized ANNs have two distinct types of architecture, the two-stage and the one-stage architecture (Lohia et. al., 2021). The former consists of a first step of extraction of Regions of Interest (RoIs) in an image and a second step including the classification and regression of RoIs, while the latter only performs the second step. The state of the art consists mostly of two-stage architectures as they perform with a much higher accuracy, but the one-stage are also utilized as they tend to be significantly faster.

The most prominent two-stage object detector architectures include the R-CNN, Fast-CNN and Faster-CNN models (Juan Du, 2018), while the one-stage architecture includes the CenterNet (Duan et. al., 2019) family model, the Yolo family model (Juan Du, 2018)and

SSD model (Wei Liu et. al., 2016). Both of these architectures include numerous other models, with a great degree of variation among them and are heavily utilized in the field of object detection.

The individual distinction between all these models, though highly technical, can be of great importance to a defect detection application, even more than in other object detection applications, depending on the size of the defect, the variations with which it is presented and its differences in appearance with the non-defective parts of the project. However, selecting an appropriate type based on these highly technical characteristics and then further customizing the model to fit the necessities of such applications, would require hours of extensive research and implementation, enough to constitute as an additional master level thesis.

The problem of determining the most suitable type of ANN, however remains and to address this, the framework, within which the model will be developed, can provide valuable information. The obvious choice for a ML framework with Deep Learning model capabilities is TensorFlow (Goldsborough, 2016). This framework, developed by Google for implementation in a wide variety of ML applications, offers the ability to utilize Deep Learning methodologies, in a far easier yet thoroughly robust way.

The fact that the TensorFlow framework has been favorited in Object Detection application, means that there is a plethora of information for the utilization of the models mentioned above, but most importantly, there are data on the characteristics of each model in relation to the utilization of each model within the framework. More specifically, there is numerical data illustrating the performance of each model, meaning the speed and accuracy of each model, within a specific dataset. This means that these metrics can be used to determine which model is more suitable for the application, at least in an initial level in regards to the speed and accuracy demands of the application and of course to prove that the overall system works both on a conceptual and on a functional level, albeit not fully optimized.

The available models of the TensorFlow framework, for object detection, are of both of the architectures mentioned above and, as mentioned, are pre-trained. The term pre-trained model, which was mentioned earlier, is rather important in the field of object detection. A pre-trained model is a saved network that was previously trained on a large dataset, typically on a large-scale image-classification task. A pretrained model can either be used as is or it can be customized to another set of objects, using transfer learning.

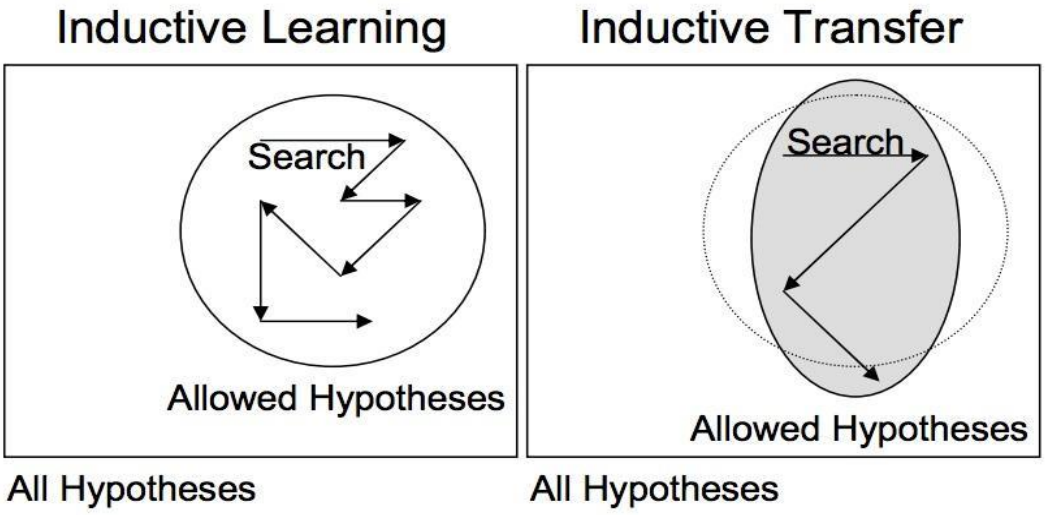


Figure 37: Transfer learning

Transfer learning (Weiss, et. al., 2016)is a ML technique where a model trained on one task is re-purposed on a second related task. This can be a potentially powerful technique in the development of a model. This form of transfer learning used in deep learning is called inductive transfer. This is where the scope of possible models (model bias) is narrowed in a beneficial way by using a model fit on a different but related task. Transfer learning is advised to be used in cases, where its implementation can result in three possible benefits:

1. Higher start. The initial skill (before refining the model) on the source model is higher than it otherwise would be.
2. Higher slope. The rate of improvement of skill during training of the source model is steeper than it otherwise would be.
3. Higher asymptote. The converged skill of the trained model is better than it otherwise would be.

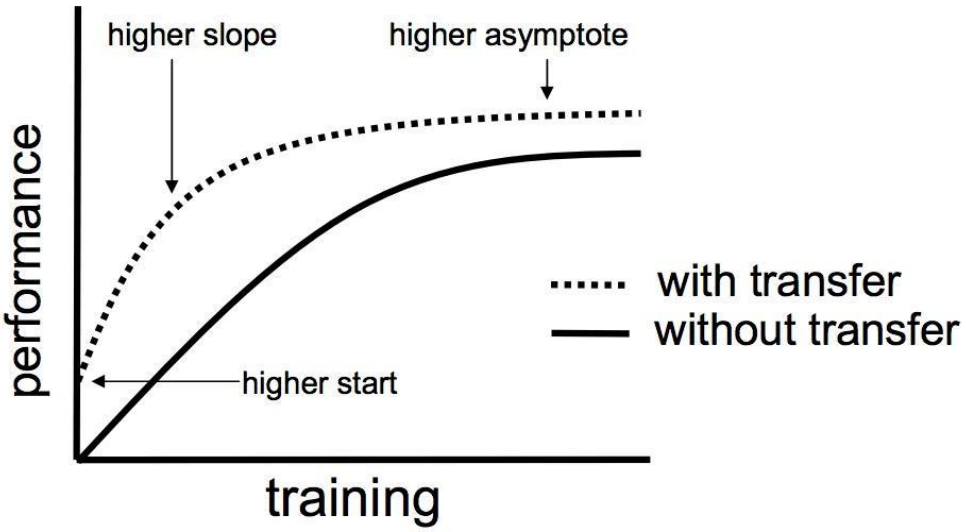


Figure 38: Benefits of transfer learning

In the specific case of object detection, significant resources would have to be used for a

customized model to be developed for a specific application and abundant data would have to be provided for training to achieve optimization. In any other case of object detection, a pretrained model modified using transfer learning is the only viable solution. As mentioned, TensorFlow offers a collection of models, referred to as Model Zoo, pretrained in the COCO 2017 dataset.

The list of available models provides the metrics required for a selection of an appropriate model. Models of different architectures and types are provided, with an indication on the size of the inputted image, and numerical values indicating the speed at which detection is accomplished and the accuracy of the detection. It should be noted that the metric for accuracy, for object detectors, is most commonly the Mean Absolute Precision, a statistical parameter derived based on the correct detection of an object detector in reference to the sum of the total correct and incorrect detections. A higher value of mAP for a specific model, correlates to a greater accuracy in object detection.

With these data now available for the provided models, a more informed decision on the most suitable one can be made. Firstly, since the main goal for defect detection within the scope of the thesis is the proof of concept, rather than optimization of the process, speed of detection is valued higher than performance. As such, two stage object detection architectures are disregarded in favor of the faster one stage architectures, meaning the CNN model family is not taken into consideration.

Furthermore, the available resources, in respect to the hardware for the creation of the model, are limited to the use of a commercially available mid-range personal computer with integrated graphics, instead of a GPU. This latter statement is an important one, as the existence of a GPU and better yet a high-end GPU, significantly decreases the time demands for the training of the model. Regardless, the training time, being the most time-consuming process of model development, meant certain compromises were necessary. Merely looking at the list of the available pre-trained model, it is apparent that for any model provided there is a list of available alternatives depending on the size of the imputed image and that the increase in size is relative to the speed of detection. What is not apparent is that the size of the image also affects the training time proportionally, meaning that increased size leads to increased training time.

At the same time, images of smaller size cannot be used indiscriminately for two reasons. To reduce the size of an image, the image needs to either be cropped or be compressed. The former option can be applied to an extent, but the region of interest in respect to the detection of defects is standard within the scope of the application and further segmentation of the images would be neither easily applicable nor time efficient, during operation of the automated station. The latter option severely affects the quality of the images, depending on the extent of the compression. This leads to the downsizing of possible defects to merely a few pixels, which makes their detection impossible even to the human eye.

Summing up the above, it becomes apparent that the most suitable pre-trained model is one that allows for pictures of a decent size, yet small enough to allow for a reasonably fast training, with the model being fast enough to run within the operation of the automated station, yet without detrimental performance in respect to its accuracy. The object detection pretrained model available by the TensorFlow framework that meets these criteria is the SSD MobileNet V2 FPNLite 320x320.

With the selection of the model complete, it is possible to proceed with the steps required to

set up an object detection ANN model within the TensorFlow framework. These steps are illustrated and thoroughly explained in the following chapters.

5.2 Selection and labeling of images

Before further explanation of the individual steps for the development and utilization of the ANN model on this application, further elaboration is required on the environment within which the TensorFlow framework will be deployed. TensorFlow is most commonly utilized with the Python programming language. It is also common for developers on Deep Learning applications and other ML projects to utilize TensorFlow and other relevant libraries, with Python on a web-based interactive computing station, called Jupyter Notebooks. The commonality of this implementation translates to an abundance of information on the development of ANN models and as such it is adopted for this application.

An additional benefit of the Jupyter Notebook station is the fact that it provides live code, equations, narrative text and visualization making it a great tool to illustrate and share code in a presentable manner with all the necessary documentation.

In the same topic, it is regarded as a safe practice to run Jupyter Notebook and install all the libraries and modules required for model development in a virtual environment. The simple reason is that a virtual environment is more easily controlled by the developer, which addresses possible issues that Python may present in respect to the management of dependencies.

Having created the virtual environment and installed all the tools necessary for this process, we can proceed with the first major stop, which is the selection of images. To train the model and validate its function as a defect detection tool, 82 high quality images were provided, of certain aluminum products with different defects or no defects at all. The reason these images were used is simply because the creation of the model was concurrent with the design and assembly of the automated station, hence it could not be used to provide the necessary images. To recreate the model creation methodology presented in the current and following chapter, the images would otherwise be collected with the control software and the electromechanical assembly of the automated station.

As mentioned in the previous chapter, this number of images would be immensely insufficient for the creation of a custom ANN model. It is, however, more than sufficient for the implementation of the transfer learning methodology that was previously illustrated. For transfer learning, the higher the number of training images, the better chances of producing an accurate object detection model. However, there is a threshold, since with more images provided, the time for the training of the model, even with transfer learning, rapidly increases. A compromise is therefore required to minimize time demand yet achieve noteworthy accuracy. For this reason, from the 82 images provided, initially 15 were used for training and 10 were used for testing within the process of creating the model and the remaining were used for manual validation of the accuracy of the model.

The images provided, as mentioned, were of extremely high quality. As it was stated in the previous chapter, the images at their current size, which is proportional to the quality, are unusable. An initial preprocessing of the images is necessary. By observing the images, it is apparent that the region of interest, in respect to defect detection, is in the middle of every

image. A script is therefore created in Python that automatically crops every picture in a specific size enough to contain the region of interest. An image cropping resulting in a new image of 995x995 pixels is more than enough to include the region of interest, while accommodating for slight differences in the center alignment between the images provided. The main benefit is that the resulting images, that will ultimately be compressed to 320x320 pixels to be inputted in the model, will include the region of interest yet only be compressed by approximately 67%, which should retain the majority of the detail of the defects to a more than acceptable degree.

Having collected the images and separated them in three categories: training, test and validation, the next step is the labeling of the objects to be detected in each of the training images. At this point, a convention must be made in regards to what constitutes a defect of a product. Depending on the type of product examined automated station and the level of surface detail required by the application, a defect can be defined differently, so a case-by-case approach is more logical. In our case study, since speed and accuracy are the main objectives, only the more visible and larger size defects will be labeled, as a proof of concept. As such, smaller surface defects such as minor cracks and pinholes are generally disregarded and only larger defects will be subjected to detection.

Regarding cast aluminum products, there is a quite large number of different parameters that can result in the appearance of surface defects. from inconsistencies in the flow of the liquid metal during pouring, to defects on the die, to fluctuation in the temperature of the die or the cooling process of the cast item and many others. Defining the cause of a surface defect on an aluminum cast item is a highly technical skill that requires both extensive knowledge on the aluminum and its casting and equally extensive experience on the actual process. As such to classify the defects based on their cause would be nigh impossible, without lengthy feedback from a skilled person.

Concurrently, a single label for defect detection could potentially pose an obstacle rather than simplify the defect detection process. The limited size of the dataset and the fact that there is a significant degree of visual difference between defects means that an attempt to categorize all defects under one label would make it difficult for the model to “define” through the training process what a defect is and how to detect it. With a large enough training data and a much lengthier training process, it would be theoretically possible for the model to reach a state where it could recognize defects despite their significant variance. This state, which in ML is referred to as Generalization, is a sought-after characteristic of a successful model. In the case of this application, however, with the limitations and constraints already mentioned, is it highly unlikely to achieve such a state for every defect present in the aluminum cast products.

A much more realistic approach would be to categorize the different defects in a much broader, visual based way. By simply observing the defects it is apparent that they generally come into two distinct shapes, albeit with variations among them. These shapes are circular for defects akin to holes and curved lines for defects akin to cracks, and combinations of these two shapes. With all the above information regarding the nature of the defects, their variety and their most basic differentiations, the most suitable classification would be according to their shape, in order to facilitate the training of the model and achieve an ANN model capable of detecting defects of different classes, even at a most basic, proof-of-concept level. By that accord, two classes of defects are created and are labeled as “Crack” and “Hole” in respect to which geometry of defect they are most similar to.

The labeling process is quite straightforward and the most common process is with the use of an open-source tool, LabelImg (Yakovlev et. al., 2020) developed by Tzutalin. It is a graphical image annotation tool developed in Python with the use of the Qt framework, with annotation on images saved on XML files in PASCAL VOC form, a format commonly used for the labeling of images for ANN applications. It has a rather intuitive layout, within which the user imports an image, creates one or more labels in respect to the individual objects that the model will be trained to detect and then assigns these labels to the objects in the image with appropriate bounding boxes. The information on the location of the selected object within the image and the label of the object is saved on the XML files previously mentioned. It is a manual process, that for a large enough dataset would be extremely laborious, but necessary for the use of supervised learning models. The environment of the LabelImg is illustrated in the image below, as is the bounding box selection tool and the list of user defined labels.

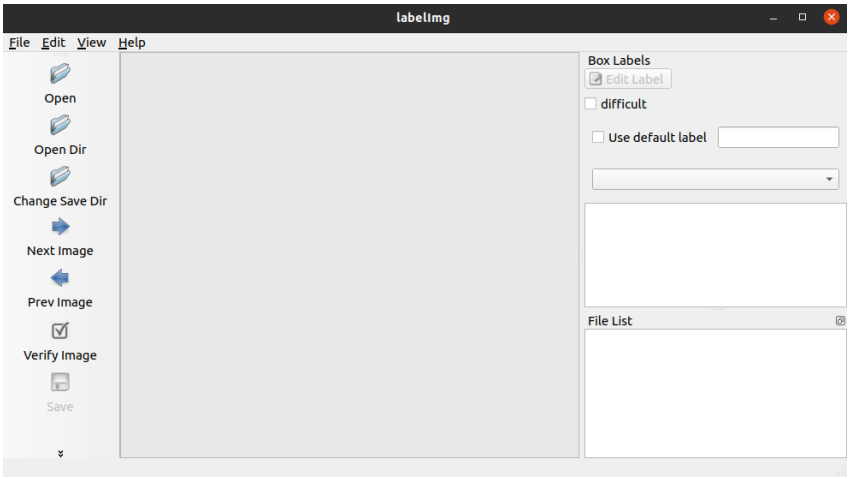


Figure 39: LabelImg image labelling environment

Though it has not yet been mentioned, the process of collecting the images, separating them in appropriate files and utilizing the LabelImg tool to label them, need to be quite structured to proceed with the training of the model. In this context structured refers to the filing of each component, meaning images and appropriate .xml files with specific names and following a strict file structure in respect to the pathway of each file inside the virtual environment previously created. Deviation from this file structure, especially without full comprehension of the model training code, will result in errors during the process. By utilizing the Jupyter Notebook, the file structure can be somewhat automated and the accompanied highly presentable documentation, within the code file, making it easy to be adapted by anyone trying to train the model for a different set of products or defects.

5.3 Training the Artificial Neural Network model

With the images collected, labeled and placed on appropriate files, it is possible to proceed with the training of the ANN model. This process is developed in Python within the Jupyter Notebook station and consists of six main steps. These steps will be illustrated within this chapter with reference to the details relevant to the training of the model, rather than the process of the development. For the latter matter, the Jupyter Notebook file containing the training code includes documentation that makes editing the process for any other set of products possible.

Firstly, as will the collection of the images, an automated process for the file structure and pathways is implemented. This process is rather critical to allow both the correct training of the model and the possible editing of the code in future applications. The setup process includes the creation of directories for the custom model, the pretrained model, all relative workspace subdirectories required for the creation of the model and finally the pipeline configuration. By pipeline, in the field of ML, it is meant the end-to-end construct that orchestrates the flow of data into, and output from, a ML model. A final check is made to ensure that the above file setup has been established and the next step can follow.

The next step of the process is the downloading and installation of the TensorFlow Model Zoo collection and the numerous required dependencies. While rather simple in its implementation, the volume of material required to be downloaded and installed on the virtual environment setup for this process, is both time consuming and in need for further inspection. A verifying script check for the correct completion of the process and for any dependency missing, manual installation must be done by the developer. The PIP package installer, which was already installed in the virtual environment, greatly aids at this process

With the dependencies installed, the next step is to create the Label Map for the training of the model, which in simple terms is the list of different labels with which the objects, in this case the defects, will be classified with. Since, in the scope of this application only one type of defect is examined, shrinkage cavities, there is only need for one label. Attention must be paid, however, that the name used for this label is the same name used during operation of the LabelImg tool for the labelling of the training images, otherwise the training process will not occur. Following this, the TFRecord files for the training and testing dataset are created. The details of the contents of these files regard to the TensorFlow framework and need no further elaboration within the context of this application.

Following, the pretrained model is reconfigured to achieve transfer learning. The update on the configuring of the pretrained model is mostly an automatic process following the configuration of the pipeline as was structured previously. The last step is the actual commands to initiate the training of the model, including the number of steps during training. In regards to the number of steps, the higher the number, the more time the training will take to be completed but during the recurring process of training the increased steps will more likely provide better results in respect to accuracy. Within the scope of this application, a number of steps equal to 2000 is deemed appropriate as anything much higher will greatly increase the required time for training, partly due to the capabilities of the hardware

Upon completion of this process, the output includes an abundance of information relevant to the performance of the model during training, both regarding the time required for detection and its performance in respect to accuracy. These metrics and all others provided by the TensorFlow framework, will be examined in the following chapter.

5.4 Evaluation of the Artificial Neural Network model

As mentioned, the evaluation of the model is possible through certain metrics that are outputted after the training of the model is complete. This information is accessible to a degree from the command prompt within which the training process runs, but it is convoluted to a degree and difficult to be extracted in a cohesive way. The TensorFlow offers a much more comprehensive way to evaluate the performance of a model, following the training

process, through a tool named TensorBoard (Mané, 2015). Amongst its many features, TensorBoard provides developers with measurements and visualizations needed to evaluate the loss and accuracy of a model.

These two metrics are the main indicators of performance of a ML model. The accuracy of the model has been mentioned and briefly explained in an earlier chapter, but for the evaluation of the model to be possible, all relevant metrics will be introduced and thoroughly explained.

As mentioned, the performance of an object detection ANN model is evaluated by a metric called Average Precision and most often by the Mean Average Precision (Anwar, 2022). It is impossible to elaborate on the meaning of this metric without introducing other metrics relevant to it and to detection and localization algorithms. Furthermore, it should be stated that the mean Average Precision (mAP) is not a metric covering the performance of a model for all of its classes simultaneously, rather than a class-by-class metric. To introduce the metrics relevant to mAP the following graph helps illustrate the relationship between them.

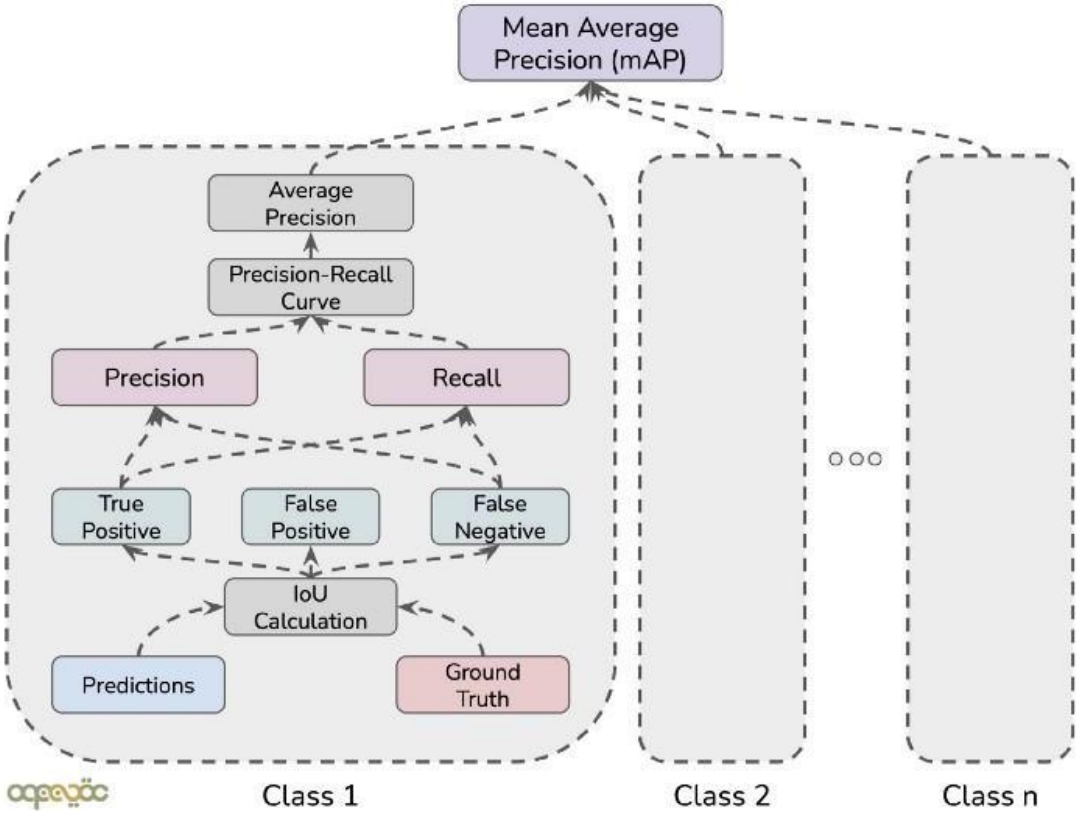


Figure 40: mAP calculation flowchart

Following the flowchart above, the introduction of each metric relevant to the evaluation of the model will be made, starting from the bottom of the chart.

1. Intersection over Union (IoU):
IoU quantifies the closeness of the two bounding boxes (ground truth and prediction). It's a value between 0 and 1. If the two bounding boxes overlap completely, then the prediction is perfect and hence the IoU is 1. On the other hand, if the two bounding boxes don’t overlap,

the IoU is 0. The IoU is calculated by taking the ratio between the area of intersection and the area of the union of two bounding boxes as shown below.

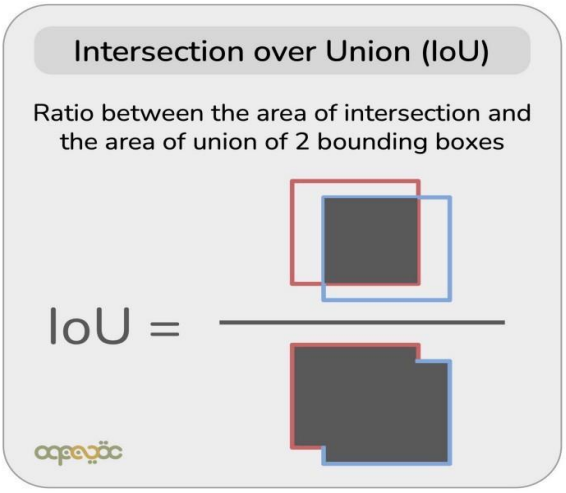


Figure 41: Intersection over Union (IoU)

2. True Positive, False Positive, False Negative:
A prediction is said to be correct if the class label of the predicted bounding box and the ground truth bounding box is the same and the IoU between them is greater than a threshold value.

Based on the IoU, threshold, and the class labels of the ground truth and the predicted bounding boxes, we calculate the following three metrics

- True Positive (TP): The model predicted that a bounding box exists at a certain position (positive) and it was correct (true)
- False Positive (FP): The model predicted that a bounding box exists at a particular position (positive) but it was wrong (false)
- False Negative (FN): The model did not predict a bounding box at a certain position (negative) and it was wrong (false) i.e., a ground truth bounding box existed at that position.
- True Negative (TN): The model did not predict a bounding box (negative) and it was correct (true). This corresponds to the background, the area without bounding boxes, and is not used to calculate the final metrics.

3. Precision, Recall
Based on the TP, FP, and FN, *for each labeled class*, we calculate two parameters: precision and recall.

- Precision: tells us how precise our model is i.e., from the total detections made by the model, how many were actually true detection, meaning in this application actual defects. Hence, it is the ratio between the true positive and the total number of defect predictions (equivalently the sum of true positive and false positive) made by the model as shown below.
- Recall: Tells us how good the model is at recalling classes from images i.e., out of total defects in the input image how many was the model able to detect. Hence, it is the ratio between the true positive and the total number of ground truth defects (equivalently the sum of true positive and false negative) made by the model as shown below.

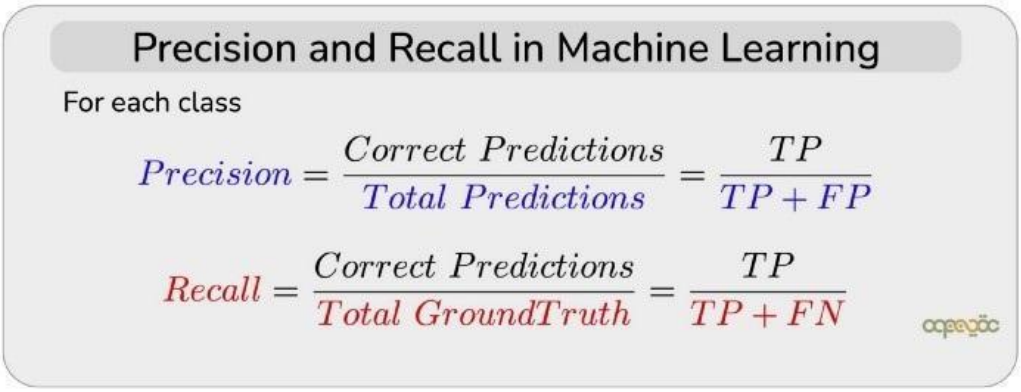


Figure 42: Precision and Recall in ML

4. Precision-Recall Curve

Ideally, we want both the precision and recall to be high i.e., whatever is detected is correct and the model can detect all the occurrences of a class. The value of precision and recall depends on how many true positives were detected by the model. Assigning a bounding box TP, FP, and FN depends on the following two things

- The predicted label compared to the ground truth label
- The IoU between the two boxes

For a multiclass classification problem, the model outputs the conditional probability that the bounding box belongs to a certain class. The greater the probability for a class, the more chances the bounding box contains that class. The probability distribution along with a user-defined threshold (between 0 to 1) value is used to classify a bounding box.

The smaller this probability confidence threshold, the higher the number of detections made by the model, and the lower the chances that the ground-truth labels were missed and hence higher the recall (Generally, but not always). On the other hand, the higher the confidence threshold, the more confident the model is in what it predicts and hence higher the precision (Generally, but not always). We want both the precision and recall to be as high as possible, hence, there exists a tradeoff between precision and recall based on the value of the confidence threshold.

A precision-recall curve plots the value of precision against recall for different confidence threshold values. With the precision-recall curve, we can see visually what confidence threshold is best for a given application.

5. Average Precision

Selecting a confidence value for your application can be hard and subjective. Average precision (AP) is a key performance indicator that tries to remove the dependency of selecting one confidence threshold value and its mathematical definition is the area underneath the Precision-Recall (PR) curve

AP summarizes the PR Curve to one scalar value. Average precision is high when both precision and recall are high, and low when either of them is low across a range of confidence threshold values. The range for AP is between 0 to 1.

$$\text{Average Precision (AP)} = \int_{r=0}^{1} p(r)dr$$

Figure 43: Equation for the calculation of the Average Precision

The actual calculation of the value of the Average Precision is possible though methods of numerical analysis, since the PR curve is drawn as connection of numerical values and is not the output of a specific function. Most common methods, especially for manual calculation is a rectangle approximation or interpolation and averaging.

6. Mean Average Precision:
AP value can be calculated for each class. The mean average precision is calculated by taking the average of AP across all the classes under consideration. i.e., $mAP = \frac{1}{k} * \sum_{i}^{k} \quad AP_i$

The process described throughs steps 1 to 6 to calculate all metrics regarding to the performance of the model is illustrated for contextual reasons, since the metrics mentioned above are outputted by the TensorFlow framework, at the end of the training of the model, through the TensorBoard tool for evaluation purposes. The mAP metrics for the training of the model, as outputted by Tensorboard are illustrated in figures 44 & 45.

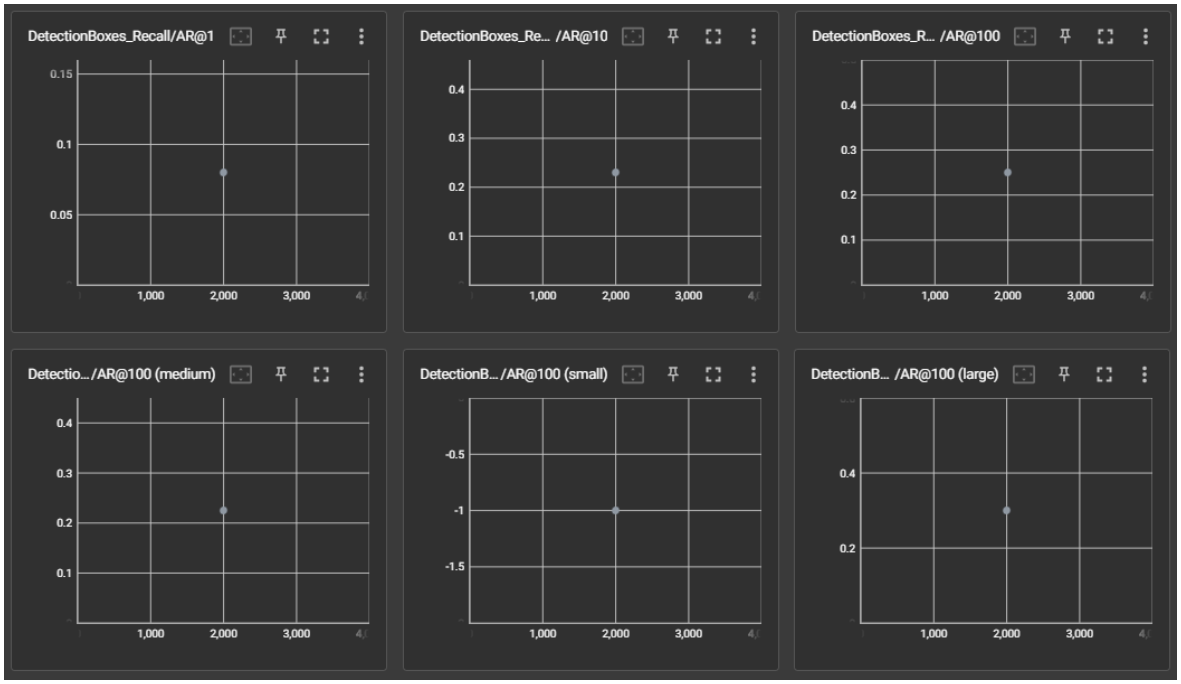


Figure 44: Recall of trained ANN

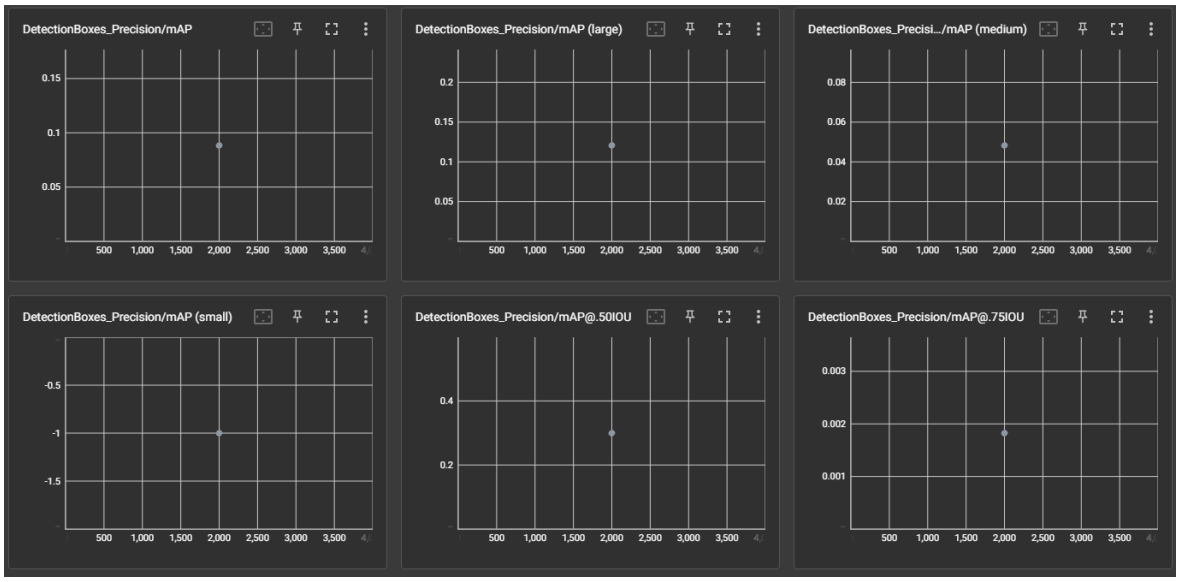


Figure 45: Precision of trained ANN

The two sets of graphs illustrated regard to the Recall and the Precision of the trained model, as these concepts have been defined above. Firstly, for both of these metrics, the horizontal axis regards to each step within the training process, a number that was inputted during the training setup of the model, that number being 2000. Only the value regarding the final step of the training process is plotted as it should represent the best recall and value attained by the model.

Regarding the recall, the six figures represent, from left to right and from top to bottom, the average recall for 1, 10 and 100 detection and the average recall of small (area<32^2 pixels), medium (32^2 pixels<area<96^2 pixels) and large objects (96^2 pixels<area<10000^2 pixels) for 100 detections. It is rather evident from these graphs that the recall of the model is stable, as the value of the Average Recall is similar for 10 and for 100 predictions, however the model is completely insufficient in recalling defects of smaller size but sufficient to a degree in the prediction of medium and larger size defects. The value for the latter defects is definitely low enough to require further optimization but it definitely establishes the capability of the model to recall classes from images

Regarding to the precision of the model, the graphs represent on the top, the precision of the model (mean Average Precision – mAP), for small, medium and large objects and on the bottom the mAP over classes averaged over IOU thresholds ranging from .5 to .95 with .05 increments, at 50% IOU and at 75% IOU. All the mAP values depicted indicate that the model is far from optimized, as it is evident by the fact the mAP value for the model on the COCO database was 0.22. Less than half of this value is reached for the trained model of this application and it is only exhibited in the larger objects, with medium objects having an even lower value and smaller objects being completely insufficient. Additionally, the higher value of mAP averaged over IOU and at 50% IOU contrary to the mAP at 75% IOU, further indicates that the model has limited capabilities in detecting the complete area of the defects Since the existence and not the complete identification of the defect is the main role of the trained model within this application, that is not a concerning matter, but definitely one that could benefit from further optimization.

The optimization of the trained model in respect to its precision and recall, will be thoroughly explained, with reference to the values of these graphs and their interpretation above, in a latter chapter.

Another metric provided by the framework is the loss functions of the model. In contrast to the other metrics mentioned above, the loss functions are calculated during the training rather than during the evaluation of the already trained model. Even so, they provide valuable feedback, since they are the only metric that provides information on the actual training process, rather than its output, being the already trained model.

Loss, or the often-interchangeable term Loss functions, is as the latter name implies a category of functions that are used extensively in the field of statistics to determine, in plain terms, how close a predicted value is to the true value. Within the field of ANNs, their role is very important as, during the training process, with every prediction (step) the loss function calculates, within a single value, the error of the prediction as defined above and provides it as an input to the next prediction in order to minimize it. This cyclic process is the core of the training of the model.

There are several loss functions used in statistical modeling and by extension ANN. They range in complexity from as simple as a Mean Absolute Error to more complex like the Categorical Cross Entropy Loss and Hinge Loss. Not every loss function is suitable for every predictive model, including ANNs and the suitability is a highly technical matter that requires both proper education on the subject and experience on the field, as both the application and the data it includes affect the final choice in loss function. Within the scope of this application the loss function for the training of the model is inherited by the pretrained model used for transfer learning.

The Tensorboard tool exports graphs illustrating the loss calculated for each step of the process, providing not only the total loss but even more specific data for loss, such as the classification loss, the localization loss and the regularization loss. The first two are self-explanatory and the latter regards the loss function on the regularization, which is a technique used in ML, to tune a function or a model by adding an additional penalty term to the error (or loss) function ultimately aiding the model to achieve generalization. The four Loss metrics, the classification, localization, regularization and the total, as exported by Tensorboard, are illustrated in figure 46.

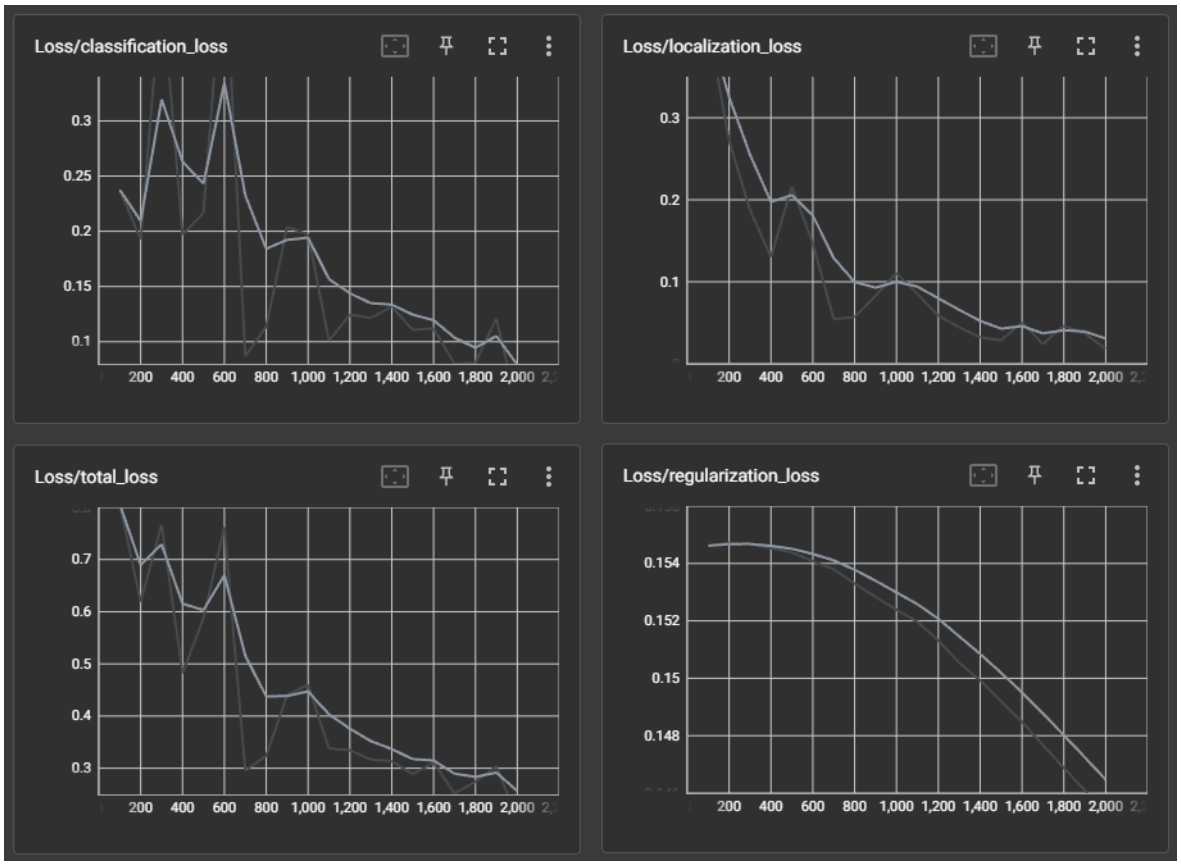


Figure 46: Loss function values during training of ANN

The most important feedback from these graphs is the trend by which the values are changing over each step of the training process. A successful ANN model, would exhibit a decline in the values of the loss functions with each step of the training, which is indeed the case for this model, but these values would eventually stabilize around a much smaller range of values. This is not the case for this model, as there is no stabilization evident in any of the loss functions. That is indicative of a model that has been trained with not enough steps. This and other parameters of the training process that would lead to a much more optimized model will be explained in a latter chapter.

5.5 Integration of Artificial Neural Network model on a GUI

There are many implementation methodologies available due to the capabilities provided by the TensorFlow framework. Since, within the scope of this application, there are no specific requirements for the deployment of the ANN model, a more general approach will be illustrated within this chapter.

In most cases, where there is camera feed within an application, the ANN model is integrated in the function of the camera feed. There, within each image received sequentially by the camera, the ANN model tries to detect the objects it has been trained to detect. With a rate of 24 frames per second, meaning a new image input every approximately 42 milliseconds, a model that runs in any time less than that can theoretically detect objects seamlessly. Why this is theoretical is because it assumes that no other functionality is undertaken by the program while the image collection and object detection run, which in any application, but the simplest ones, is not true.

The result is quite visible in systems with hardware of relatively low specifications, as any

attempt to integrate the ANN model to a slightly complex GUI and control software results in intense lagging, to the extent that it renders the whole application inoperable. Referring back to the relevant chapter on the GUI and the way it operates, any lagging introduced to the loop can inadvertently cause the GUI to crash and the application to terminate unexpectedly. Unexpected crash, within the scope of this application, requires a reboot of the control software with the added time consumption for a full homing operation, as the position of the camera within the working area of the automated station is not saved without explicit command from the user or during normal shut down process. For all these reasons, an unoptimized ANN model integrated in the control software, running simultaneously with the camera feed, is deemed unnecessarily risky. Such integration can be easily accomplished once optimization of the model is complete and use of high specification hardware is possible.

To avoid the risks mentioned above and since there is no immediate demand by the application for live detection of defects, during every moment of operation of the automated station, an alternative approach has been implemented. An additional GUI was created, within which the ANN trained model has been integrated. The process for the creation of the additional GUI is no different in concept than the one described in chapter 4 of the present thesis, but significantly simpler. The only elements required in the Defect Detection GUI are:

- a loading button to allow the user to load an image taken by the camera using the "Photo" button of the control software GUI or any other external image file
- a cropping button that allows the user to focus the detection process on a specific part of the image, thus allowing for better accuracy of the prediction
- and lastly, a detection button to initiate the defect detection process by the model.

Following the process illustrated in an earlier chapter, the GUI is created in Qt Designer, exported, converted in python script format, the buttons are connected to their appropriate functions and a dark theme is applied to the GUI. The functions for the loading of the file and the cropping of the image are quite simple to be implemented in Python and will not be further explained. In regard to the implementation of the detection function, the image path is inputted, the image file subjected to detection is accessed using the OpenCV library, then the image is converted into a matrix using the NumPy library and the resulting matrix is converted to form recognized by the model. The detection is then initiated and which outputs the bounding boxes and the labels. These two elements are then placed on the original image, which is then plotted using the Matplotlib library. The GUI with an example of an image with defects detected is illustrated in figure 47.

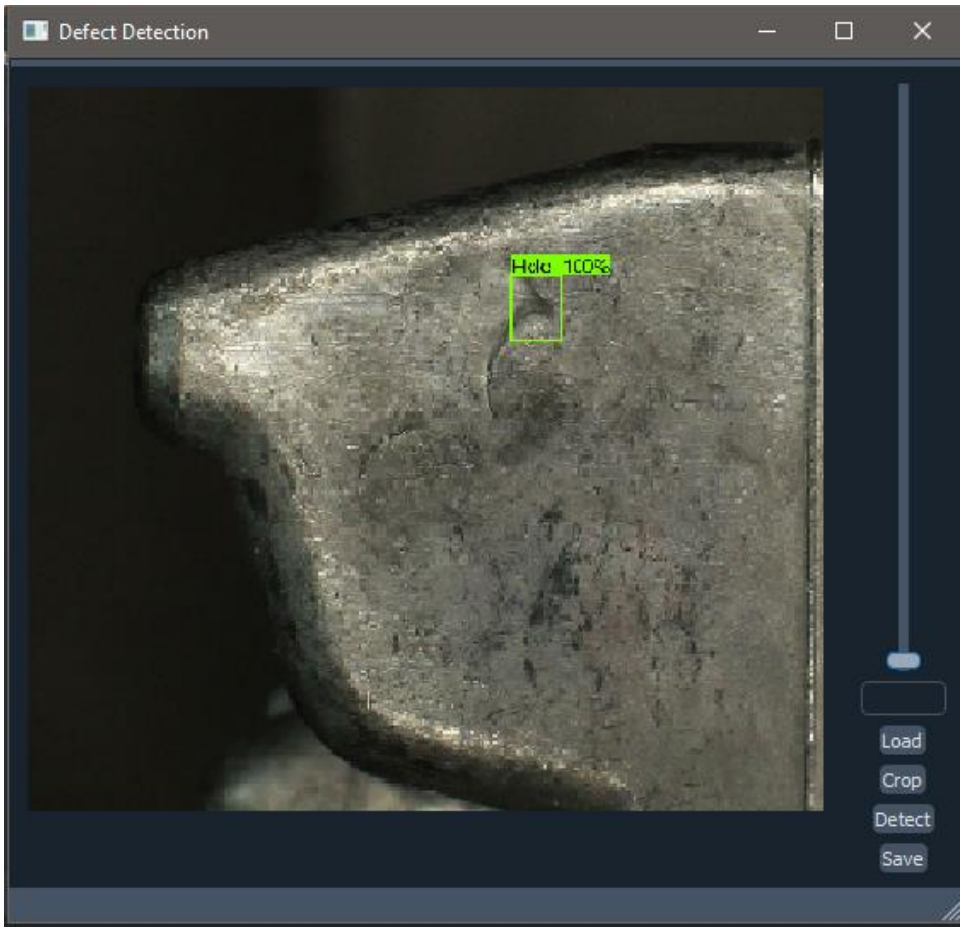


Figure 47: GUI for defect detection

6. Conclusions and suggestions for further research

With the completion of the design and construction of the automated station, the development of the control software and the training, evaluation and implementation of the ANN model for defect detection, we can proceed with the individual examination of the capabilities and possible flaws of each component and the overall system. In continuation to the sequence by which the individual parts of the system were presented in the previous chapters, the conclusions for each subsystem have been separated in regards to the physical subsystem, meaning the electromechanical assembly, the control software and the defect detection ANN model. Each subsystem will be examined in the following chapters.

6.1 Functionality of automated station

In regard to the mechanical system, following the assembly of all mechanical parts, no issues have been presented to indicate the necessity for any change, major or minor. A few suggestions can be made nevertheless.

As mentioned in the relative chapter, the automated station, and more specifically its frame, is secured by 90-degree corner brackets, fixed in place using bolts and T-nuts. At the moment, no issue has risen with the integrity of the construction. However, the existence of moving parts on the frame and the resulting vibrations from the acceleration and deceleration of these parts, can potentially be problematic in the future. These vibrations can cause the bolted connection to loosen in time, thus allowing a degree of relative motion, even in the millimeter scale, between the individual parts of the frame. This relative motion can cause anything from increased vibration of the frame to misalignment of the axis. The former could range from an inconvenience to the user to further loss of fixture in bolted connections. The latter, depending on the severity, can cause loss of accuracy to the movement of the camera to increase and put stress on other mechanical components of the assembly even to the extent of resulting in failure of a mechanical part.

The potential issues of the bolted connections can be easily resolved with the use of chemical adhesive agents, commonly referred to as thread lockers, that spread between the threads of the bolt and the nut, bonding them together in a much more secure manner. In addition to this, it is highly recommended that aluminum extrusions are placed diagonally in the lateral sides of the statical frame that are not used for access to the working area of the automated station. This recommendation aims to increase the stability of the overall construction, such that even in a case were some of the bolted connections are loose enough to allow for relative motion, the diagonal aluminum extrusions resist any relative motion, thus preventing the overall frame from being in any way slanted or misaligned.

Figure 48: Proposed design with diagonal aluminum extrusions

In regard to the electrical system, no issues have risen during the operation of the automated station. Any suggestions made would have to be relative to the ease of use of the system rather than the performance of its functionality. One such suggestion would be the use of a microcontroller with Wi-Fi capabilities, such as the ESP32 microcontroller, thus allowing the user to operate the automated station remotely and without a computer directly connected to the overall system.

Another suggestion, which has already been scheduled to be integrated but exceeded the scope of this thesis due to time constraints, is the installation of a light source. A light source would be greatly beneficial to the quality of the images taken by the camera. Additionally, images of the same item taken with a light source from different angles could result in better detection.

In the case of a single light source, the implementation is quite simple, with a relay being connected to one of the existing microcontroller's free digital pins. The relay would have to be intermediate between the power line to the light source and be of the Normal Open (NO) type. Upon a relative command from the user through the GUI the microcontroller would signal the relay to close, thus allowing the light source to be fully powered. In this implementation no control over the intensity of the light source is possible through the GUI.

In the case of multiple light sources in the perimeter of the frame or capability to control the intensity of one or more light sources, a second microcontroller would be required, as the first one has most digital pins used by other existing components. The two microcontrollers would have to be in communication, in order for the GUI to pass commands to the first microcontroller for translation and then to the second microcontroller for light control execution. For connection of a larger number of light sources to a microcontroller, a number greater than the digital pins of the second microcontroller, a multiplexing connection, such as charlieplexing, would have to be implemented.

6.2 Functionality of control software

The functionality of the control software has also been without any issues. Any further alterations or additions would serve to increase the ease of use, the aesthetics and perhaps the performance and efficiency of the GUI script, the latter being a matter of optimization within the scope of software development rather than a need that has risen from the operation of the GUI. At this point, since no such necessities have risen, no further elaboration on the optimizing of the control software code will be made within this chapter, as it would only be generic and unrelated to the objective of the application.

As mentioned, with proper optimization of the trained ANN model, it would be possible and beneficial for the scope of this application, to integrate the defect detection model to the camera feed. It has already been thoroughly explained why only an optimized model would be recommended for integration, but even such a model needs only run when necessary to further facilitate a risk-free operation of the control software GUI.

It is thus proposed to create a button that enables the model to run defect detection straight from the camera feed. This capability can be turned off during movements that occur outside a region of interest and be enabled once the camera is close enough to a region with potential defects. This process can be done manually by the operator of the control software. The only difference in implementation of the model in such a way with the way it implemented and illustrated on a previous chapter, is that instead of an image file being imported from a file path, processed an inputted to the model, the image file now comes directly from the camera feed with the use of the OpenCV library. All next steps of image processing, detection and update of the image with bounding boxes and labels remains the same, in terms of development.

As mentioned in the relative chapter, the implementation of the control software regarding the functionality of the microcontroller is also remarkably successful within the scope of the application. A more common implementation would be the use of the GRBL Arduino Uno Shield, with the Arduino, the latter being uploaded with a GCode interpreter. That implementation would require most of the available memory of the Arduino Uno. As common an implementation as it may be, the limited available memory left for any other task is often the cause of issues in the correct execution of commands by the microcontroller and as a result an improper functionality of the system, in this case being the automated station. With the use of a microcontroller with more available memory, this implementation would perhaps be preferred for its popularity and much easier implementation, but would still not be a perfect match for this application or nearly as suitable as the one implemented.

6.3 Performance of defect detection

The performance of the model has been thoroughly explained using the appropriate metrics on the chapter illustrating the evaluation process of the trained ANN model. As a proof of concept, it has been shown to be capable of detecting defects on aluminum products, but at its current performance level, it is not very probable that it would actually be deployed in a real-life application. However, certain changes in key parameters of the training of the model, could have resulted in a much higher accuracy.

Firstly, in respect to the hardware used, a setup with a GPU is deemed necessary for such applications. Having completed the successful training of a model with the current

specification, which lacks a GPU, it has become apparent that the training process is not only slower but even likely to fail, using only the CPU. This is a rather important observation, as many decisions made regarding the training parameters of the ANN model, were based around the minimizing of the time required for training, simply because a more extensive training could very realistically result in the CPU reaching its very limits for an extended period of time. In regard to the GPU required for such an application, there is really no limit, with the only suggestion being the use of one with a VRAM of at least 4Gb.

Having established such a hardware setup, the most important training parameter that can increase the accuracy of the model is the number of the images used in training. Again, there is, theoretically, no limit to this number as the higher it is the better the model will become in detecting defects. Realistically, however, the time it would take to label these images manually would be a major obstacle, let alone the training process, which would be exponentially increased regardless of the hardware used. The use of transfer learning significantly brought down the number of images required for training and there is no specific rule of thumb for this number, however, based on the literature and suggestions from developers implementing ANN models with the process illustrated in the current thesis, the use of around 100 images for the training of each class would be ideal for this application.

Having mentioned the classes of the ANN, it should also be noted that further examination into the correct distinction of the classes must be made. For the trained ANN model of chapter 5, the classes were defined in the most intuitive way possible, based on their shape i.e., long thin lines being cracks and rounded dents being holes. Further examination of a much larger training sample might reveal a more intuitive or better suited way to classify defects. A better classification, one that provides the most differentiation between the object of each class among all objects or lack thereof, will undoubtedly result in a more accurate ANN model.

At the presentation of evaluation metrics of the trained model and more specifically of the output of the loss functions, it was noted that loss is reduced during the iterations of the training process and eventually its output stabilizes around a certain value. Since in the metrics regarding the output of the loss function for the trained ANN model of this application, the value output did not show any sign of stabilization and was still on a downward trend when the training was complete, is it quite apparent that more iterations of the training process were required. These iterations, referred to as steps, were selected to be 2000. The small number of steps lead to a model prone to overfitting (Ying, 2018), meaning one trained to an extend on the training dataset, but highly ineffective on any other similar dataset. A better approach would have been the use of a bigger training dataset, which would then be separated into smaller groups of images (batches) and training process consisting of multiple passes through the entire dataset (epochs), This would result in a much more optimized model, one far less prone to overfitting.

Lastly, should the implementations mentioned above do not result in high enough accuracy, a more drastic solution would be the use of an ANN model of different architecture. As mentioned in a previous chapter, the architecture of an ANN model can significantly impact its performance both in speed and in accuracy, with these two characteristics generally being inversely proportional. In case the speed of detection within an application is deemed secondary to the accuracy of the model, a different architecture than the one chosen in this application would have to be implemented. The SSD model used was a one stage detector, with relatively low-quality image input, resulting in a fast but generally low accuracy detection. For better accuracy, a model from the CNN family, with an image input of at least

640x640 pixels, would be much more preferable, with a more likely candidate being the Faster-CNN, which has a respectable speed of detection. The creation of a model, with a more customized architecture and training, without transfer learning, from a dataset of at least 10000 images per class would ideally result in the greatest possible accuracy and speed of detection, however, its development and implementation would take immense resources and is therefore not recommended in any but the most demanding applications.

References

1. Advanced Connection Systems - JST catalogue Vol. 120e. (n.d.). Retrieved from https://www.jst.fr/doc/jst/pdf/jst-connector-catalogue-vol-120e.pdf

2. Al-Saffar et. al. (2017). *Review of deep convolution neural network in image classification.* Jakarta, Indonesia: IEEE. doi:10.1109/ICRAMET.2017.8253139

3. Amridesvar et al. (2020). *Modeling phase distribution in build platform for*. IOP Publishing. doi:10.1088/1757-899X/988/1/012047

4. Anand, Sheila. (2019). *A Guide for Machine Vision in Quality Control.* CRC Press. doi:10.1201/9781003002826

5. Anwar, A. (2022, 5 13). *TowardsDataScience.com*. Retrieved from https://towardsdatascience.com/what-is-average-precision-in-object-detection-localization-algorithms-and-how-to-calculate-it-3f330efe697b

6. Arief Wisnu Wardhana et. al. (2019). *Stepper motor control with DRV 8825 driver based on square wave signal from AVR microcontroller timer.* AIP Conference Proceedings 2094, 020015 (2019. doi:10.1063/1.5097484

7. Atraszkiewicz et. al. (2020). *Frictional Behaviour of Composite Anodized Layers on Aluminium Alloys.* MDPI. doi:10.3390/ma13173747

8. Badamasi, Y. A. (2014). *The Working Principle Of An Arduino.* Abuja, Nigeria: IEEE. doi:10.1109/ICECCO.2014.6997578

9. Baluta, G. (2007). *Microstepping Mode for Stepper Motor Control.* Iasi, Romania: IEEE. doi:10.1109/ISSCS.2007.4292799

10. Berger, C. (2018). *Conception de la structure d'une machine de vision par ordinateur.* National Technical Univeristy of Athens, Manufacturing Technology Devision. Athens, Greece: National Technical University of Athens.

11. Culjak et. al. (2012). *A brief introduction to OpenCV*. Opatija, Croatia: IEEE. doi:CD:978-953-233-072-4

12. Duan et. al. (2019). *CenterNet: Keypoint Triplets for Object Detection.* Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV).

13. Euro-bearings.com. (n.d.). *Euro-bearings.com.* Retrieved 10 2022, from https://www.euro-bearings.com/bushingsgeneral.html

14. Goldsborough, P. (2016). *A Tour of TensorFlow.* Technische Universität München, Fakultät für Informatik. doi:10.48550/arXiv.1610.01178

15. Hanson Technology - GX Aviation Connector Datasheet. (n.d.). Retrieved from https://www.handsontec.com/dataspecs/connector/GX16.pdf

16. Hasan et. al. (2018). *Implementation and Manufacturing of a 3-Axes Plotter Machine by Arduino and CNC Shield.* Al-Najaf, Iraq: IEEE.

doi:10.1109/IICETA.2018.8458071

17. Hechtel, K. (2021). *PLASTIC MATERIALS FOR FRICTION.* Curbell Plastics, Inc.

18. Jianglin Huang. (2015). *An empirical analysis of data preprocessing for machine learning-based software cost estimation.* Information and Software Technology. doi:10.1016/j.infsof.2015.07.004

19. Juan Du. (2018). *Understanding of Object Detection Based on CNN Family and YOLO.* Hong Kong: IOP Publishing Ltd. doi:10.1088/1742-6596/1004/1/012029

20. Kaiji Sato et al. (1995). *Control and Elimination of Lead Screw Backlash for Ultra-Precision Positioning.* JSME International Journal. doi:10.1299/jsmec1993.38.36

21. Knörig et. al. (2009). *Fritzing: a tool for advancing electronic prototyping for designers.* Potsdam, Germany: TEI '09: Proceedings of the 3rd International Conference on Tangible and Embedded Interaction. doi:10.1145/1517664.1517735

22. LeCun et. al. (2015). *Deep Learning.* Macmillan Publishers Limited. doi:10.1038/nature14539

23. Locke, D. (2008). *Guide to the wiring regulations: IEE wiring regulations (BS 7671: 2008).* John Wiley & Sons.

24. Lohia et. al. (2021). *Bibliometric Analysis of One-stage and Two-stage Object.* Library Philosophy and.

25. Long et. al. (2017). *Accurate Object Localization in Remote Sensing Images Based on Convolutional Neural Networks.* IEEE. doi:10.1109/TGRS.2016.2645610

26. Mané, D. (2015). *"TensorBoard: TensorFlow's visualization toolkit, 2015.".*

27. Mezei. (2017). *Cross-platform GUI for educational microcomputer designed in Qt.* IEEE East-West Design & Test Symposium (EWDTS). doi:10.1109/EWDTS.2017.8110109

28. Prodanov et. al. (2022). *Reliability of low-power stepper motor drivers.* Department of Electronics, Faculty of Electrical Engineering and Electronics, Gabrovo, Bulgaria. doi: 10.1109/ET55967.2022.9920214

29. Schneider Electronics. (n.d.). *Nema 17 - Datasheet.* Retrieved from https://datasheetspdf.com/pdf-file/1260602/Schneider/NEMA17/1

30. Smid, P. (2000). *CNC Programming Handbook: A Comprehensive Guide to Practical CNC Programming.* Industrial Press Inc.

31. Van der Walt et. al. (2019). *PLA as a suitable 3D printing thermoplastic for use in external beam radiotherapy.* Australas Phys Eng Sci Med. doi:10.1007/s13246-019-00818-6

32. VCalc. (2022). *https://www.vcalc.com/wiki/vCollections/Leadscrew-Torque-lift*. Retrieved 10 2022, from https://www.vcalc.com/wiki/vCollections/Leadscrew-Torque-lift

33. Venkat Sai. (2017). *A Critical Review on Casting Types and Defects.* Telangana, India: Print ISSN. doi:10.32628/IJSRSET1732150

34. Wei Liu et. al. (2016). *SSD: Single Shot MultiBox Detector*. Lecture Notes in Computer Science(), vol 9905. Springer, Cham. doi:10.1007/978-3-319-46448-0_2

35. Weiss, et. al. (2016). *A survey of transfer learning.* J Big Data. doi:10.1186/s40537-016-0043-6

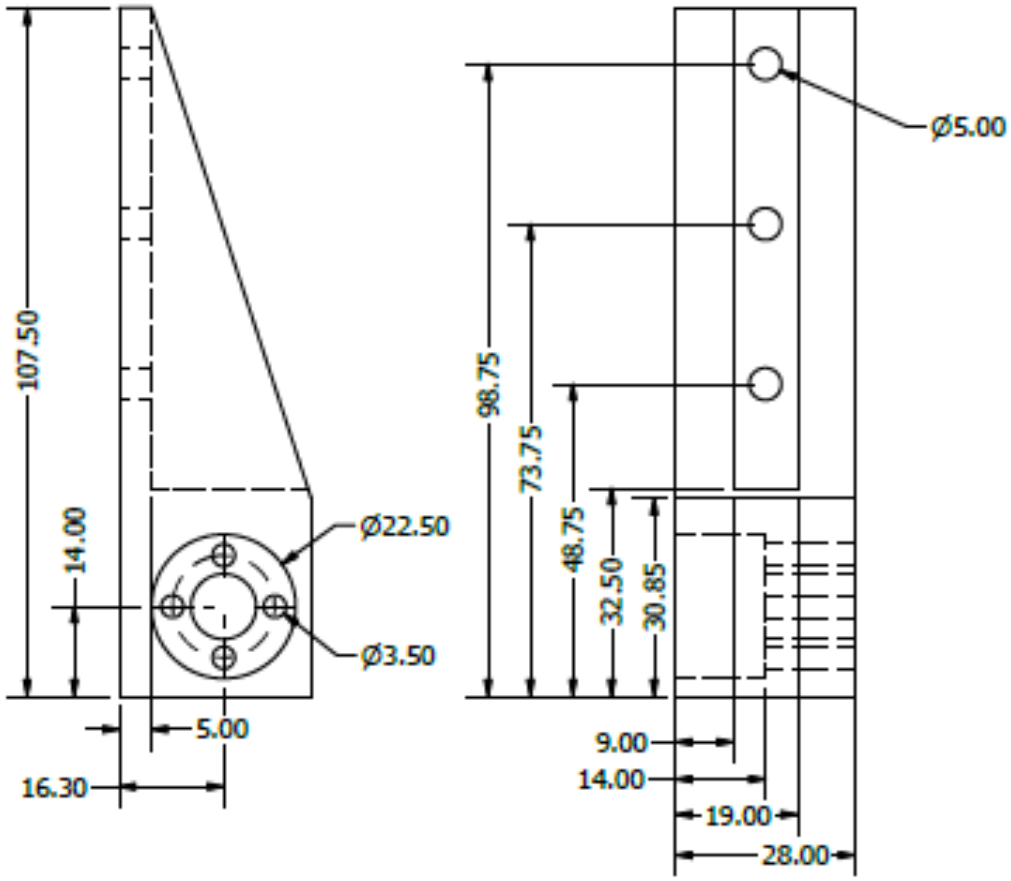
36. Willman. (2022). *Beginning PyQt: A Hands-on Approach to GUI Programming with PyQt6* (2nd Edition ed.). Sunnyvale, CA, USA: Apress Berkeley, CA. doi:10.1007/978-1-4842-7999-1

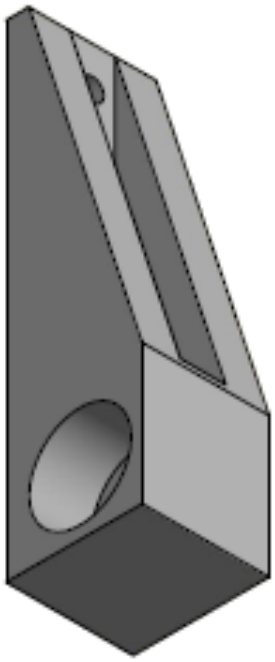
37. Yakovlev et. al. (2020). *AN APPROACH FOR IMAGE ANNOTATION AUTOMATIZATION FOR ARTIFICIAL INTELLIGENCE MODELS LEARNING.* doi:10.20535/1560-8956.36.2020.209755

38. Ying, X. (2018). *An Overview of Overfitting and its Solutions.* IOP Publishing Ltd. doi:10.1088/1742-6596/1168/2/022022

39. Zhong-Qiu Zhao et. al. (2019). *Object Detection With Deep Learning: A Review.* IEEE. doi:10.1109/TNNLS.2018.2876865

Annex A – CAD Designs of 3D printed parts





Part 1: Connection part between X-axis motion system and Y-axis sub-assembly.