



Budapest University of Technology and Economics

Gergő Bendegúz Békési

Vision System Development for Thermal Transient Measurement Automation of DBCs Using Fully Convolutional Deep Learning Neural Networks, Image Processing Algorithms and Tree-structured Parzen Estimator

Scientific Student Associations' Report

MENTORS:

Dr. Péter Ekler

Dr. Ágnes Urbin

Szilárd Szőke

BUDAPEST, 2022

Abstract

Due to the many trends of the 21st century, such as the rise of devices using modern power electronics, the fast electrification of the automotive industry and the rapid increase in electric energy consumption, it is expected that the technology of direct bonded copper substrates will continue to play a key role within the electronics industry, because they have a prominent role in technological innovation. Due to the mentioned trends, these devices must cope with an increasing power dissipation density, which exposes them to a significant thermal load. Planning for this load is often a safety-critical task, because direct bonded copper substrates are installed in applications such as servomotors or automatic gearboxes, where failure during operation is particularly dangerous. In addition, temperature is the leading cause of failure in this industry, accounting for more than half of all failures, well ahead of vibration and moisture. In the research and development phase of direct bonded copper substrates, the thermal test can be performed with thermal transient measurement, T3Ster measurement. To create high-quality thermal models of electronic devices, certain model identification processes should be robotized. This reduces the effect of the human error factor and increases the number of measurements that can be performed in a unit of time. That is why the automation of the T3Ster measurement has significant added value both from an economic and technical point of view, as it increases competitiveness and reproducibility. This automation saves manual sample preparation for each component, thus reducing the number of human working hours and enabling measurements without human supervision. It is advisable to support the described process with a vision system, because with the help of this, the measuring robot receives information about the deviation of the individual electronic devices to be measured from the ideal position and orientation in the measurement space. In this way, the robot can correct this displacement, and as a result, the safety-critical measurement process cannot be interrupted. Due to the small size of the electronic devices, the demand placed on the vision system is high-precision navigation. This requires at least 0.25 mm accurate position calculation and 1° accurate orientation calculation.

I contributed to this automation by developing a vision system in the Thermal Identification Team of Robert Bosch Ltd. I have developed a complex system, of which first task is to create a high-quality picture containing sufficient information. In the absence of this, it is difficult to develop a good image processing algorithm. A good optical system can be a more effective starting point for the entire vision system. For this, I chose optics and sensor considering the smallest size to be observed, the expected object distance range and the angle of view, and in the most emphatic part of my work, I developed software for processing the image information required for automation. Through a server-client communication, I supported the three-degree-of-freedom TTT robot with a vision system software made in Python 3 language. It uses several machine learning algorithms and fully convolutional neural networks based on deep learning. I chose task-specific parameters for the former, and I developed the latter myself. Fine-tuning the hyperparameters was a key task of my work. With the help of the Tree-structured Parzen Estimator, I determined appropriate settings for the categorical, continuous and discrete parameters of the deep learning-based semantic segmentation neural network. This is how I chose a suitable optimization algorithm, dilation, batch size, channel number, depth, epoch number, weight decay, learning rate, and learning rate decrease rate and decrease step interval. I created the dataset required for training and validation with the selected optics and sensor used for measurements and annotated them by hand. This means a total of 422 images, which I used with 80% - 20% training-validation split with additional data augmentation. I trained and evaluated network architectures 12149 times and achieved 98.37% IoU accuracy with the best configuration. Thanks to this and the SIFT algorithm integrated into the workflow, I created a robust vision system suitable for navigation with an accuracy of 100 μm and 0.75°. In addition to the fact that this fulfils the requirements of the task, during my literature investigations, I could not find any work that would have implemented semantic segmentation and navigation with such high accuracy in the case of direct bonded copper substrates. So, my work can be used for new industrial projects and developments in the future. Furthermore, the combined use of the Tree-structured Parzen Estimator, the fully convolutional semantic segmentation neural network and the SIFT algorithm, which helped to achieve the presented results, is a new approach.

Table of Contents

Abstract	i
Table of Contents	ii
1 Introduction.....	3
2 DBCs and their thermal characterization	5
2.1 Correlation between the reliability of electronic systems and temperature	5
2.2 Temperature measurement	7
2.2.1 Use of thermocouples	7
2.2.2 Use of thermal imaging cameras.....	7
2.2.3 Thermal transient measurement, use of T3Ster [16].....	7
3 Development of thermal transient measurement technique from the perspective of automation.....	13
3.1 Development of measurement methodology.....	13
3.1.1 Manual sample with soldering and wires.....	13
3.1.2 Use of a pin bed.....	14
3.1.3 Building a robot	15
4 Developing and selecting the necessary tools and hardware for the vision system.	17
4.1 Selection of sensor and optics [24]	17
4.1.1 Focal length calculation.....	18
4.1.2 Selection of the camera system	19
4.2 Hardware resources and mechanical design.....	22
5 Software implementation	24
5.1 Transmitting camera feed from Raspberry Pi server to client machines.....	24
5.2 Configuration of the vision system.....	24
5.2.1 Calibration of the distortion of the optical system	24
5.2.2 Calibration of the position and orientation of the optical system [41]	26
5.3 Displacement determination.....	28
5.4 Semantic segmentation.....	28
5.4.1 Literature summary.....	29
5.4.2 Implementation.....	31
5.5 Keypoint detection [131][132]	44
5.6 Displacement calculation	45
5.6.1 Shift correction	45
5.6.2 Rotation correction	45
5.7 Evaluation of results.....	46
5.7.1 International aspects of semantic segmentation.....	47
5.7.2 Examination of accuracy.....	47
6 Summary and opportunities for further development	52
7 References.....	53

1 Introduction

“You insist that there is something that a machine cannot do. If you will tell me precisely what it is that a machine cannot do, then I can always make a machine which will do just that.” (János Neumann)

The aim of my scientific student research project is to implement a vision-based robot navigation system. This task contributes to the automation of a widely used measurement in the electronics industry, the T3Ster measurement. With the integration of optics, mechanical engineering, electronics, and computer control, the system presented in this work is able to replace an 8-10 hour human work process with a 15-minute configuration and the subsequent automated measurement. This work process involves the thermal transient measurement of direct bonded copper (DBC) ceramic substrates. These devices are used in applications such as servo motors or automatic transmissions, where failures during operation can be very dangerous, making their testing and evaluation particularly important.

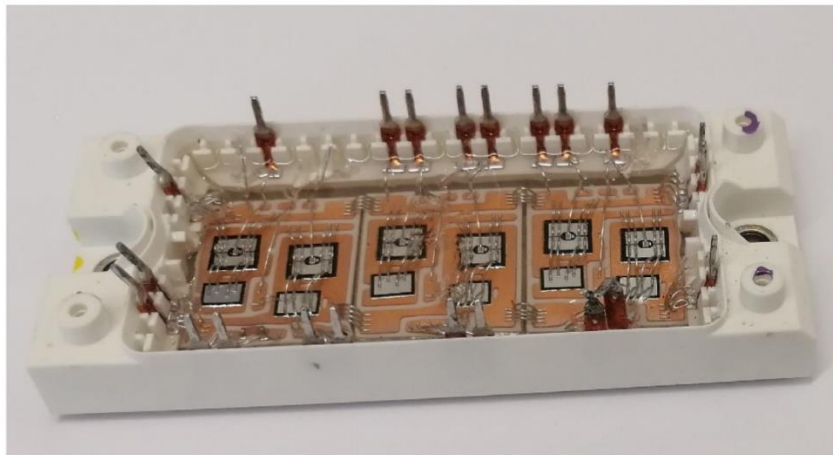


Figure 1: Illustration of a power module using DBC technology [1]

My thesis is structured as follows: After this introduction, Chapter 2 provides an overview of the relevance and theoretical foundations of the measurement to be automated, while Chapter 3 discusses the measurement challenges and the need for and process of automation. Automation can be broken down into hardware and software problem solving. Chapter 4 deals with hardware challenges, such as selecting the appropriate camera system, i.e. the appropriate sensor and optics. In my thesis, I determined the minimum pixel size that meets the task-specific accuracy requirements and chose an architecture that fits the system's physical characteristics. Based on this, I describe the economically optimal sensor-optics pair.

The task of the vision system is not only to create images but also to process them. As described in Chapter 5, I perform the radial and perspective distortion calibration of the camera. Subsequently, my developed software performs semantic segmentation on the images of the individual examined devices, so that, by using the keypoints of the segmented parts, it can calculate displacement. The semantic segmentation is carried out using fully convolutional neural networks. For this, I did a comprehensive literature review in Chapter 5.4.1. The hyperparameter-optimization of the problem-specific network architecture using Tree-structured Parzen Estimator is described in Chapter 5.4.2.9, while the estimation with an IoU value of 98.37%, which can be considered very good even on an international level, is evaluated in Chapter 5.4.2.10. Originally, the entire system was expected to achieve accuracy of 0.25 mm displacement and 1-degree rotation, but I managed to achieve accuracy of 0.1 mm displacement and 0.75-degree rotation, which demonstrates the effective functioning of the selected hardware and developed software in Chapter 5.7. Finally, in Chapter 6, I summarize the achieved results and outline the directions for further development.

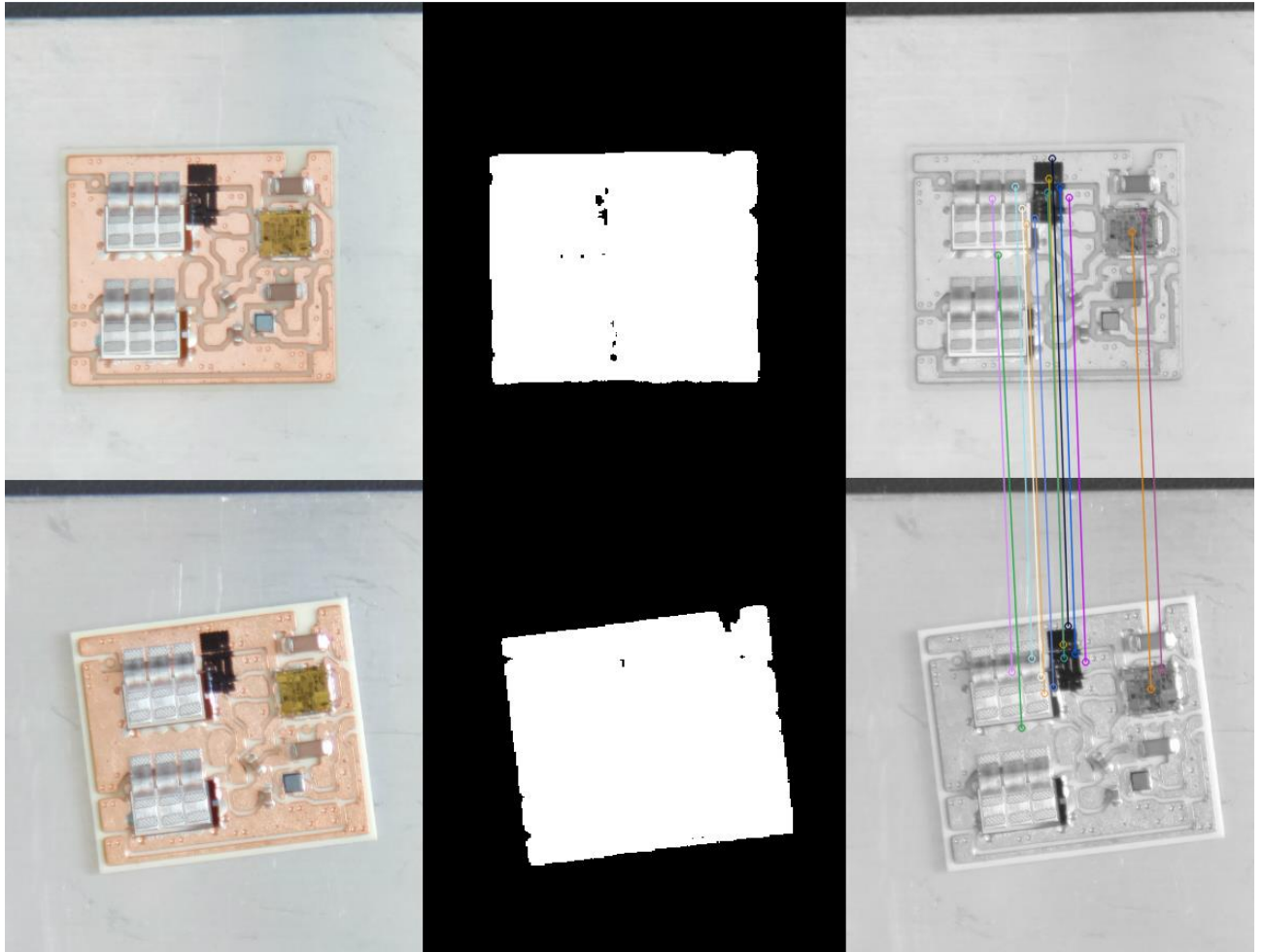


Figure 2: Illustration of supporting robot navigation with image processing

2 DBCs and their thermal characterization

Currently, the global market for DBCs is experiencing continuous growth. The reason for this is that they play a crucial role in technological innovation, and they are essential for satisfying the constantly increasing consumer demands in the field of electronic devices. All indications suggest that significant growth can be expected in the next 5-6 years in the area of active electronic devices, semiconductors, and therefore carriers, typically located on these ceramic substrates [2-11]. The cited global trends make it clear that research and development in the field of DBCs and the devices located on them are particularly important for future technologies. New innovative industrial solutions often require smaller active devices, smaller transistor sizes, or higher performance. By integrating more of these elements per unit area or using higher-performance ones, the power dissipation density increases. Therefore, special attention must be paid to the thermal testing of these devices. In the following, I will explain the need for automated measurement and introduce the theoretical basis for it. The so-called T3Ster measurement provides a solution to the problem described above, which I made faster and more efficient with my developed electronic device during my work. Thermal transient measurement is a scientifically exciting and challenging area both in terms of physical description and measurement techniques. In this field, I was able to replace approximately 8-10 hours of active human work with a 15-minute configuration followed by automated measurement.

2.1 Correlation between the reliability of electronic systems and temperature

To develop reliable electronic devices, it is essential to be aware of the main causes of their failures. Figure 3 provides information on this.

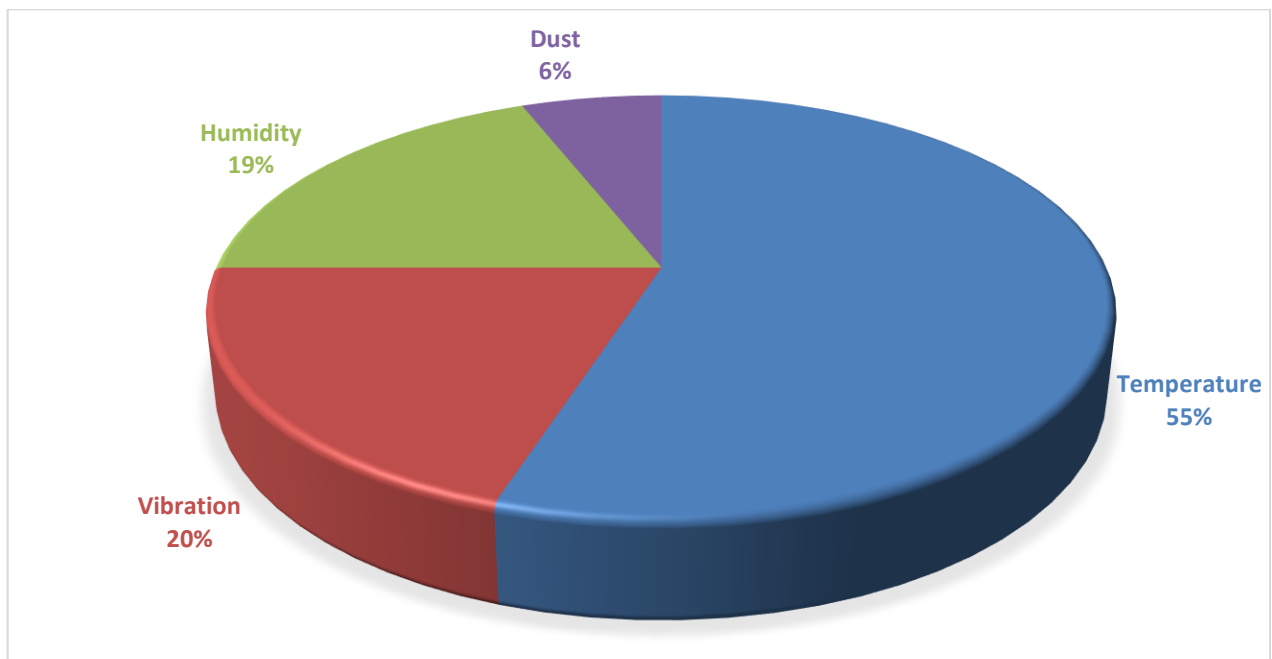


Figure 3: Main causes of electronic device failures [12]

Based on these, it can be seen that the most common cause of failure for electronic devices is temperature, which causes more than half of the failures [12]. Vibration, humidity, moisture, various pollutants, and dust are significantly less common causes. Furthermore, Lemmens et al. [13] and Wintrich et al. [14] unanimously draw attention to the fact that the expected lifetime of direct-bonded copper (DBC) ceramic carriers is significantly shorter at higher temperatures. These devices are tested in the industry using two-stage cyclic tests, including cyclic power tests. The first stage is a longer heating stage, during which the device reaches its maximum temperature. The second stage is a shorter cooling stage, during which the tested sample cools down to the minimum temperature. The difference between the minimum and maximum temperature is the amplitude. The specific parameters of the entire process are included in the SEMIKRON standard [14]. Figure 2 draws attention to the temperature dependence of the number of cycles until failure. It can be seen that the lifetime of a DBC exponentially decreases with an increase in average temperature, and the effect of amplitude increase also causes nearly exponential lifetime reduction at constant average temperature.

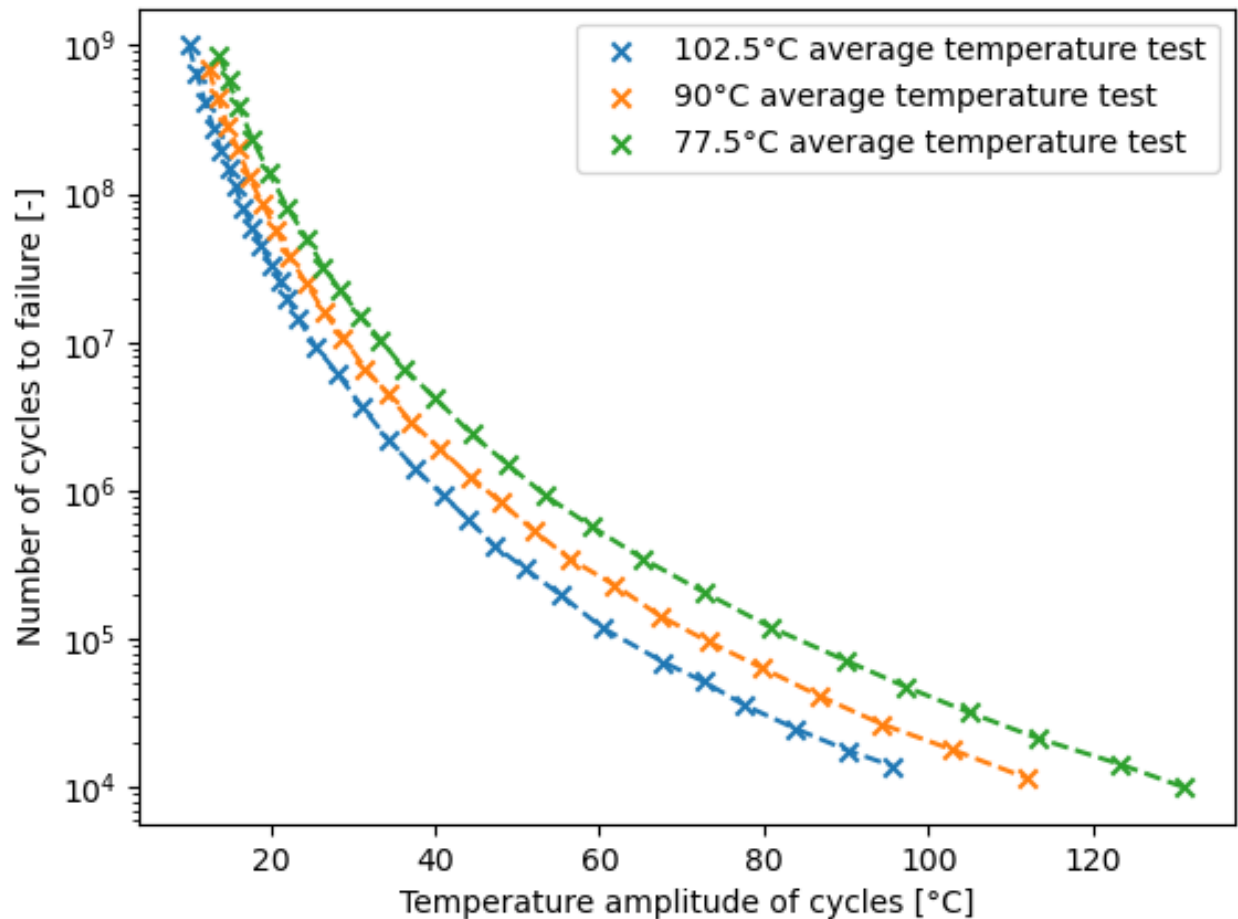


Figure 4: Temperature dependence of DBC failure [14]

Therefore, for the sake of reliability, as well as for the amount of material used in production, the economics and sustainability are very important factors, that the DBCs and the devices placed on them must be within the appropriate temperature range during operation, while fulfilling their tasks in the entire system. To

determine this range and to develop devices that can operate properly in this range, the heat input that they are subjected to must be measured and quantified in some way.

2.2 Temperature measurement

There are also some traditional methods for measuring the temperature of DBCs and the electronic components mounted on them. Their advantage is that they are relatively inexpensive and simple solutions, but they do not provide information on the internal temperature of individual active components or chip temperature. From this perspective, a better choice may be to use thermal transient measurement, which also provides information on these parameters [15-17].

2.2.1 Use of thermocouples

One simple solution can be the use of thermocouples. By electrically connecting two different metals, typically welded, placing the junction on the surface to be measured, a temperature gradient is created along the different metals, resulting in an electrical voltage between them. The measured voltage depends on the material quality of the two wires and the temperature difference between the junction and the measurement location. By knowing the characteristic of a specific wire pair, the voltage measurement can be converted into a temperature measurement.

2.2.2 Use of thermal imaging cameras

Another possible method is the use of a thermal camera, which takes advantage of the fact that every material emits electromagnetic radiation due to the thermal motion of its charged particles. This radiation can be converted into an image by the sensor of the thermal camera, and different temperature surfaces can be displayed with different colors. However, this solution is also not satisfactory for surface measurement, just like the use of a thermocouple, as it does not provide information about the internal temperature of the electronic devices placed on the DBC.

2.2.3 Thermal transient measurement, use of T3Ster [16]

The T3Ster [17] (Thermal Transient Tester) measurement, complemented with appropriate software support [18], provides indirect information on how temperature changes over time in the entire internal structure of the device. Therefore, understanding its operation requires knowledge of the analogy between electrical and thermal systems.

2.2.3.1 *Analogy between electrical and thermal systems [19]*

The analogy between electrical and thermal systems can be demonstrated by describing the similarity between an elementary transmission line segment and an elementary volume element that allows only one-dimensional energy flow.

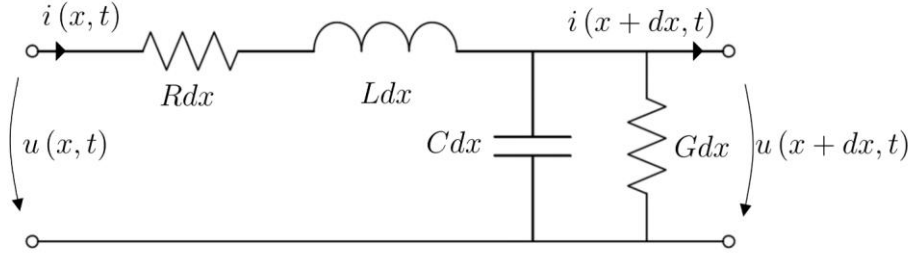


Figure 5: Elementary transmission line model

The spatial derivative of the voltage and current of an elementary transmission line segment can be written as follows [20]:

$$\frac{\partial}{\partial x} u(x, t) = -Ri(x, t) - L \frac{\partial}{\partial t} i(x, t) \quad (1)$$

$$\frac{\partial}{\partial x} i(x, t) = Gu(x, t) - C \frac{\partial}{\partial t} u(x, t) \quad (2)$$

where based on Figure 5:

- u is the voltage,
- x is the spatial parameter,
- t is the time,
- R is the resistance,
- i is the current,
- L is the inductance,
- G is the conductance,
- C is the capacitance.

Distributed RC transmission lines are often used in electronic calculations, which means that boundary conditions of $L = 0$ and $G = 0$ can be applied. Based on these conditions:

$$\frac{\partial}{\partial x} u(x, t) = -Ri(x, t) \quad (3)$$

$$\frac{\partial}{\partial x} i(x, t) = -C \frac{\partial}{\partial t} u(x, t) \quad (4)$$

Therefore, the model is significantly simplified:

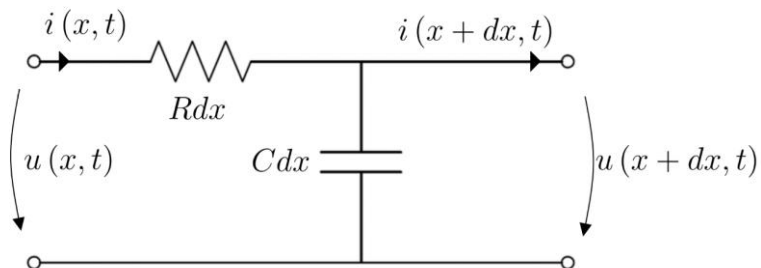


Figure 6: Elementary RC transmission line model

In the case of thermal systems, it is convenient to start with the principle of conservation of energy. Then, the difference between the incoming and outgoing heat flow can be written in the x-direction [21] as:

$$\dot{Q}(x) - \dot{Q}(x + dx) = \frac{\partial}{\partial x} \left(-\lambda \frac{\partial}{\partial x} T(x, t) dydz \right) dx \quad (5)$$

where based on Figure 7:

$\dot{Q}$ is the heat flow,
 x, y, z are the spatial parameters,
 λ is the thermal conductivity,
 t is the time,
 T is the temperature.

As heat conduction in semiconductors is typically approximated as one-dimensional, it is not necessary to write this equation in the y and z directions.

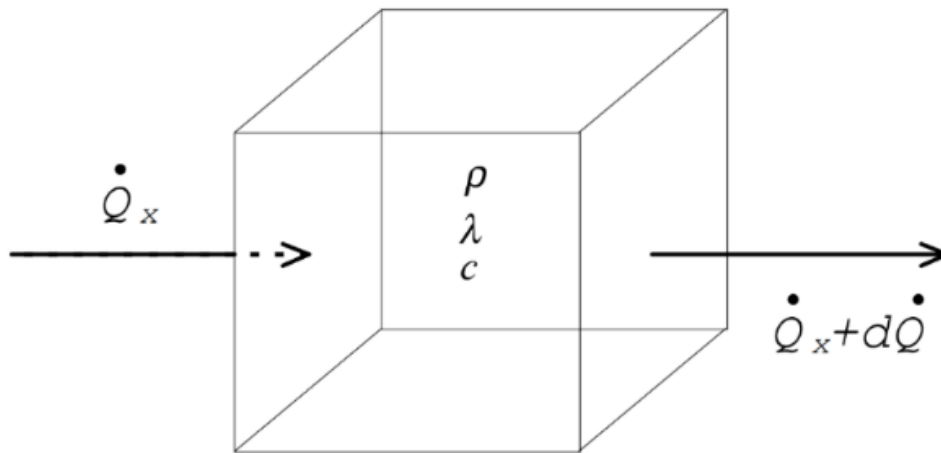


Figure 7: Elementary volume element model

By writing the energy balance equation of the volume element dV , assuming that it is source-free (i.e., no energy is generated), it can be stated that the excess incoming energy equals the change in enthalpy, i.e.:

$$-\frac{\partial}{\partial x} \left(-\lambda \frac{\partial}{\partial x} T(x, t) \right) dxdydz = \rho c_p dxdydz \frac{\partial}{\partial t} T(x, t) \quad (6)$$

where:

ρ is the density,
 c_p is the specific heat capacity at constant pressure.

Here, if $dV \neq 0$ and the temperature dependence of λ is negligible:

$$\lambda \frac{\partial^2}{\partial x^2} T(x, t) = \rho c_p \frac{\partial}{\partial t} T(x, t) \quad (7)$$

In this case, the second-type boundary condition can be used, which states that the heat flux density $\dot{q}$ is known at the boundary of the dV :

$$\dot{q}(x, t) = -\lambda \frac{\partial}{\partial x} T(x, t) \quad (8)$$

Thus, based on equations (7) and (8):

$$\frac{\partial}{\partial x} T(x, t) = -\frac{1}{\lambda} \dot{q}(x, t) \quad (9)$$

$$\frac{\partial}{\partial x} \dot{q}(x, t) = -\rho c_p \frac{\partial}{\partial t} T(x, t) \quad (10)$$

Thus, the similarity between equations (3), (4), and (9), (10) becomes apparent. The analogy is evident. The temperature difference can be compared to the potential difference, the reciprocal of the thermal conductivity to the resistance, the heat flux to the current density, and the product of the density and the isobaric specific heat to the capacitance. Therefore, thermal problems can be reduced to the analysis of RC models.

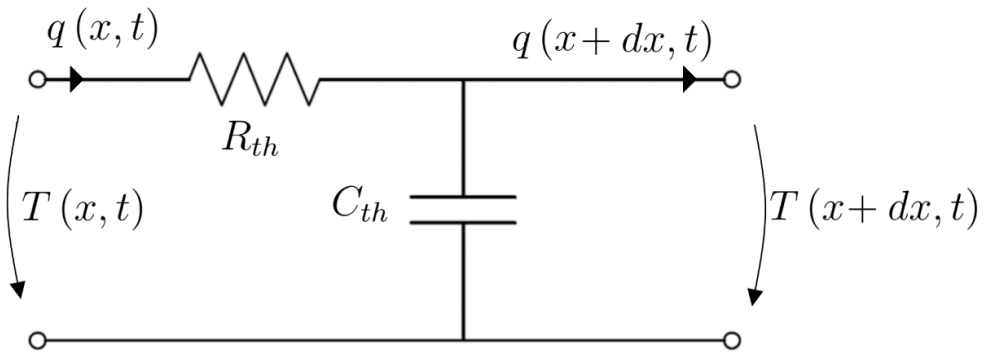


Figure 8: Thermal RC model

2.2.3.2 Analysis of distributed parameter systems [16]

The RC networks presented above can be used to model thermal systems related to electronic devices. As mentioned in connection with Figure 7, thermal conduction is often considered one-dimensional and neglected in the other directions. This is advantageous, as it allows for the analysis of each layer inside the electronic device that differs thermally. It is important to emphasize that this is a distributed system, meaning that the different layers located in space have different thermal resistance and thermal capacitance. Consequently, in a distributed parameter system, each RC element will have a different time constant. In this case, the system can be described by the so-called Foster model or Cauer model.

Mathematical basis of the measurement [22]

Such a distributed parameter RC system gives a weight function, or Green's function, in response to a Dirac delta excitation. From this, the response function can be derived for any excitation using the convolution integral:

$$T(t) = \int_0^{\infty} W(y)P(t-y)dy \quad (11)$$

so:

$$T(t) = W(t) \otimes P(t) \quad (12)$$

where:

T is the temperature,
 t is the time,
 W is the weight function,
 P is the excitation function,
 y is the variable.

Actually, the weight function that characterizes the system is not known. This is what needs to be determined, so deconvolution, rather than convolution, is necessary, as shown in (13):

$$W(t) = T(t) \otimes^{-1} P(t) \quad (13)$$

However, the Dirac delta excitation is very difficult to achieve in practice. Therefore, a better solution may be the much easier to implement unit step function, for which the response is the unit step response function, as shown in equation (14):

$$a(t) = W(t) \otimes h(t) \quad (14)$$

where:

a is the unit step response function,
 h is the unit step function.

So:

$$a(t) = \int_{-\infty}^{\infty} W(y)h(t-y)dy \quad (15)$$

$$a(t) = \int_0^{\infty} W(y)dy \quad (16)$$

Where from:

$$W(t) = \frac{d}{dt} a(t) \quad (17)$$

So the main advantage of this method over measuring the response to the Dirac delta weight function is that the unit step excitation is easier to implement. This is a better solution, but in practice, it may be useful to work not with the unit step function itself, but with its transform. The reason for this is that in the case of the unit step, maintaining constant power after the "step" is not a simple task. More precisely, the function $h_1 - h(t)$ can be used, where h_1 is a constant equal to $h(\tau)$, for $\tau > 0$. This function, which could also be called the "unit step down" function, is illustrated in Figure 9.

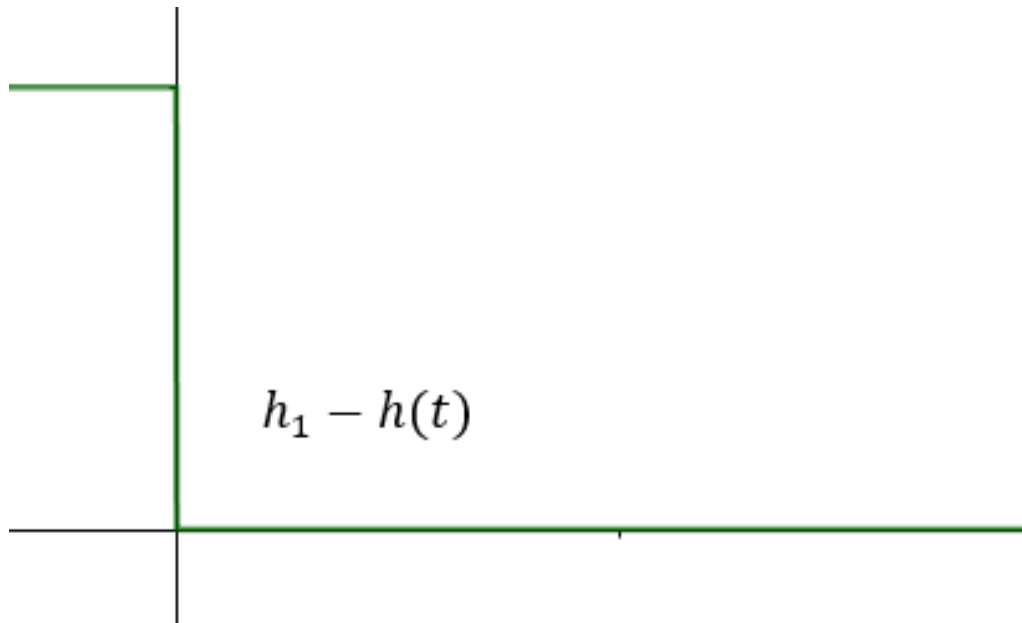


Figure 9: „Unit step down” function

In summary, when characterizing electronic devices using a T3Ster device, it is necessary to observe the decreasing temperature as the response when a constant power value is turned off. The response of the cooling device is measured based on electrical properties. All semiconductor devices have several so-called TSPs (temperature-sensitive parameters), which are temperature-sensitive parameters. For a simple diode, this is, for example, the forward voltage. The T3Ster measures this voltage with high accuracy and sampling, and the temperature is calculated based on this data through software. Therefore, it can be said that the characteristics of the investigated RC system can be determined based on the measured response $a(t)$. The distribution of thermal resistances and heat capacities within the entire DBC device, that is, the chip and the entire device structure, can thus be examined [23].

3 Development of thermal transient measurement technique from the perspective of automation

In the introduction, as I mentioned, my team conducted the thermal characterization measurement, as presented in Chapter 2.2.3. As I write in Chapter 2.1, this is an engineering activity that requires both economic and technological forward-looking and intensive development. To increase competitiveness and innovation, automation of processes is essential, not only in manufacturing but also in research and development. To create modern, high-quality models of individual electronic DBC devices for certain product development, it is worthwhile to robotize certain model identification processes. This reduces human error factors and increases the number of measurements that can be performed in a unit of time. This increases the added value and makes the process more economical.

3.1 Development of measurement methodology

We, in accordance with the description in the introduction of Chapter 3, are constantly striving to renew ourselves and create competitive products with innovative technologies. This includes the automation of our thermal transient measurements. The process and necessary electronic device, the T3Ster, was developed at the Department of Electronic Devices at the Faculty of Electrical Engineering and Informatics at Budapest University of Technology and Economics. Today, this method is used worldwide for non-destructive structural analysis and thermal optimization of integrated circuits and discrete components. However, to the best of my knowledge, when writing this work, no industrial participant had yet developed a computer vision-based project to automate the various DBC investigations in this measurement, neither domestically nor abroad. Therefore, even in the field of research and development, this work is considered pioneering. In the following points, I will present the thought process and steps involved in this development.

3.1.1 Manual sample with soldering and wires

The first method we used for measuring DBCs was a solution that required continuous active human work and presence. We strengthened the wires onto the individual components by soldering and heated the individual devices through them, then measured their response signal. Of course, we did this with the help of T3Ster. The small and often hard-to-reach transistor legs on the DBC made sample preparation quite cumbersome. Care had to be taken to ensure that the electrical conductor of the solder did not come into contact with the parts on the ground potential during the measurement, and, of course, the solder material of the wires of two electrically insulated legs could not contact each other. Here, of course, the preparation time is difficult to calculate, as it is difficult to solder the given wires to the given small legs. And even after this was successful, particular attention had to be paid to preventing any unwanted short circuits between any two materials, and to avoid any mechanical damage.

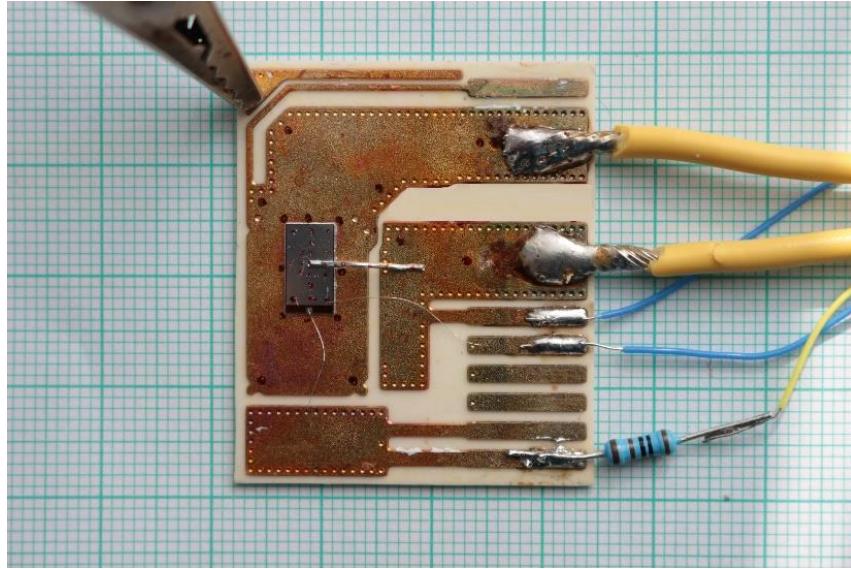


Figure 10: Manually prepared DBC

Preparation for each DBC required a time-consuming and exhausting manual process. We thought about finding a way to do this more efficiently, automating the process. It was important to avoid the time-consuming soldering required for each sample. Moreover, we wanted to increase our competitiveness in the long run by measuring more samples in the same amount of time. Thus, we developed a method that reduced the sample preparation time required for each device.

3.1.2 Use of a pin bed

The essence of the method mentioned at the end of section 3.1.1 is to avoid the need to solder wires onto the legs of every new sample and check for contacts each time. This led to the development of the pin bed shown in Figure 11.

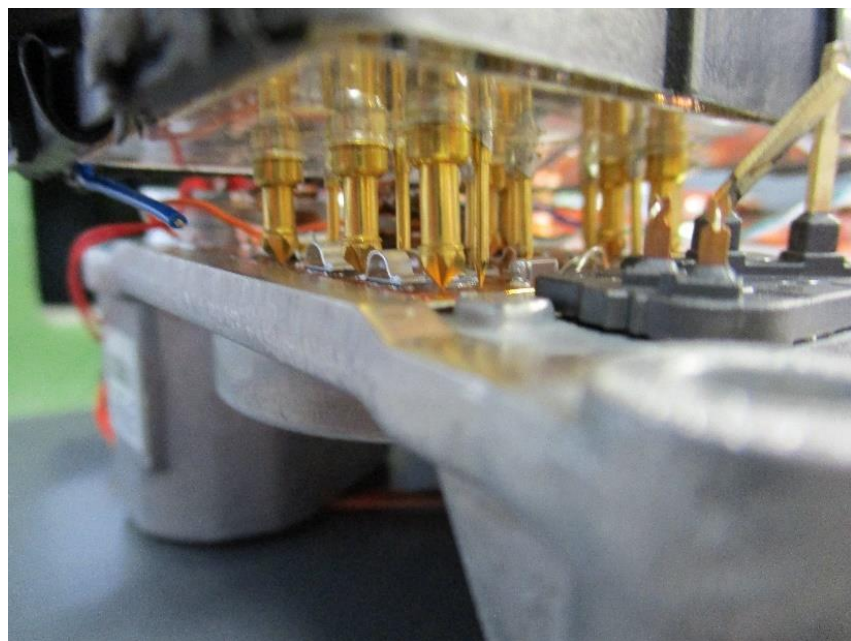


Figure 11: Use of a pin bed

Using electrically conductive pins, contact can be made with the devices to be measured from above with the help of the pin bed. As a result, the time-consuming process of preparing each sample is only required for the pin bed, and this time is distributed among all the samples to be measured. Overall, the gross measurement time for each sample is reduced. However, even this method still requires human presence because when a measurement is completed, the sample must be replaced under the pins. This also takes time and does not allow for unsupervised measurements at night, for example. Therefore, we have reached the point where we aimed to develop a robotic system.

3.1.3 Building a robot

This intelligent system consists of an automated measuring robot and its associated hardware and software framework, which can be configured quickly. The robot can be seen in Figure 12.

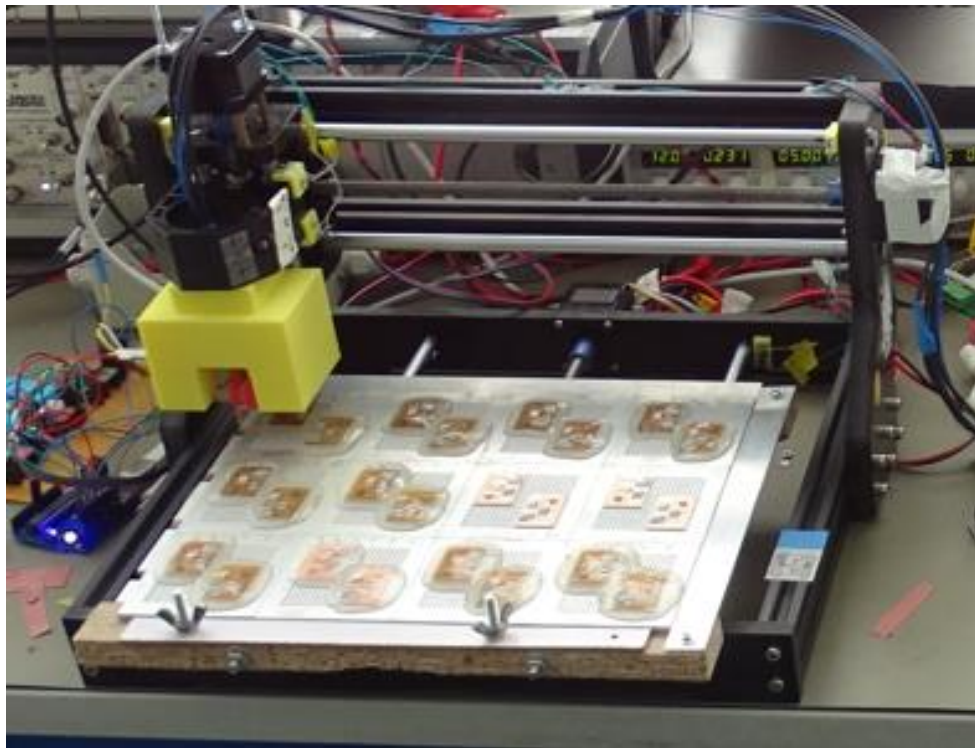


Figure 12: Measurement with the help of the robot

The robot only needs to be set up above the first DBC, which is part of the configuration, and then the system will operate automatically. The entire setup is implemented using a modified CNC 3018 Pro milling machine. This was accomplished by replacing the engraving motor and tool with a measuring pin bed. The robot is controlled using a state machine model in LabVIEW environment. Here, the robot calculates how many steps its stepping motors need to take in which direction, which are capable of executing various movements with a precision of 100 μm . The control could seem simple with fixed DBC locations, but due to the different sizes of the various types, this is not possible, and the accuracy of placing the individual DBCs to the aluminium sheet is also a complicating factor because the measuring pins make contact with legs that can be as narrow

as a few tenths of a millimetre. Therefore, a system is needed that can intelligently and precisely provide input to the robot regarding the location of the measuring instruments. The goal is to achieve an accuracy of at least 250 μm . I took on the task of developing a vision system that can provide the LabVIEW code with correction data.

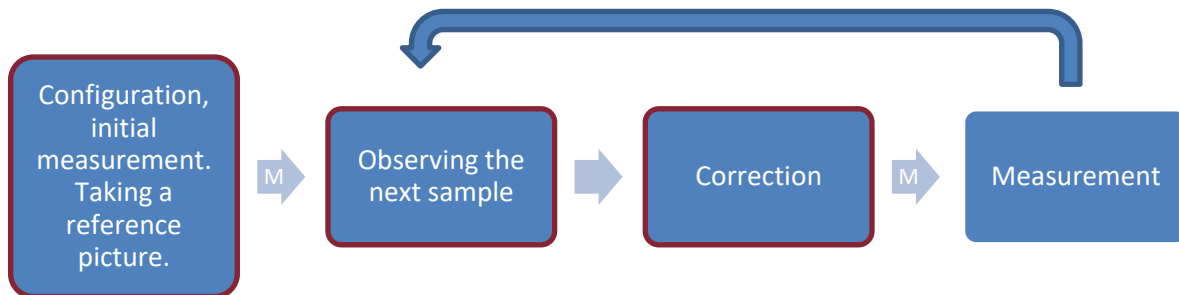


Figure 13: Process of thermal testing

As shown in Figure 13, the vision system influences significant steps in the robot control process, indicated by the red-framed steps. The robot's movements are represented by arrows labelled "M". Configuration takes place before the first measurement. At this point, the measuring pin bed is manually positioned in contact with the first DBC. From here, the system operates autonomously. After successfully completing the first measurement, the robot moves with a fixed offset to take a photo of a well-measured reference object. Then, cyclically, it positions the camera over the next sample and takes a picture of it. Based on the comparison of these two images, the software I developed determines the necessary position correction relative to the previously well-measured sample. The robot performs this correction movement beyond the offset and then measures the sample.

4 Developing and selecting the necessary tools and hardware for the vision system.

My task was to create a system that can calculate displacement between two DBC target objects based on camera images. As the system serves to automate the measurement of very small components, the minimum accuracy requirement was 250 μm . Achieving a stepper motor accuracy of 100 μm would be a great, long-lasting, and robust solution. Currently, the robot pin bed cannot rotate around its vertical axis, but in the long run, this feature is also a goal due to the possibility of DBCs being misaligned in this way. Therefore, the vision system must take this parameter into account as well. In the next section, I will detail the selection of the sensor, optics, and hardware resources necessary to meet the technical requirements mentioned above.

4.1 Selection of sensor and optics [24]

To select the appropriate components for the vision system, it is useful to summarize the starting parameters. The viewing angle of the observed object can be estimated as zero, as the offset mentioned in section 3.1.3 allows the robot to bring the camera directly above the DBCs and stop there. Thus, the individual DBCs will always be located along the optical axis of the vision system. It is advisable to choose the smallest size to be observed as the desired accuracy, assuming the best achievable result of 100 μm . The object distance is not a fixed value, there may be an additional heat exchanger placed under the currently measured device, or possibly a body equipped with additional cooling plates. As an initial estimate, I assumed the object distance to be between 20 and 100 mm. The former may only be necessary in the case of a very large heat exchanger, but it is the confident underestimation of the object distance. The latter can only occur if the robot views the workspace from a completely elevated, distant position. Therefore, by selecting these limits, I aimed for a flexible, multi-functional, and robust solution that satisfies many requirements. Similarly to choosing the smallest size to be observed, it is also a good decision to choose a smaller pixel size, such as approximately 2.5 μm , due to the small objects to be observed. This also complicates the hardware selection just like the wide object distance range. The latter requires the focal length of the optics to be adjustable over a wide range, while the former eliminates many sensors from the options available. Therefore, the calculation have to be performed by an iterative approach. If there is no really good solution for the given pixel size, it can be slightly modified. The summary of the quantities discussed here is presented in Table 1.

Table 1: Starting parameters for sensor and optics selection

Quantity	Symbol	Value, range of values
Viewing angle	φ	0°
Object distance	s	[−100 mm; −20 mm]
Minimum observable size	x	0,1 mm
Pixel size	px	2,5 μm

4.1.1 Focal length calculation

It is important to specify which pixel size to consider when selecting the pixel size. A typical principle in camera selection is to choose the length of one side of the rectangular pixel if the object is primarily fixed in a horizontal or vertical orientation. The pixel size in the direction perpendicular to the smallest observable image details is crucial. However, if the object orientation is not fixed, it is advisable to consider the pixel diagonal as the characteristic size for safety reasons. The same principle should be applied in this application as well, as mentioned in the introduction of Chapter 4, since detection of rotated objects is also necessary [25].

In general, if the goal is to capture images of objects, it is reasonable to represent the smallest size with at least 2 pixels. Defining an edge or a boundary can already work this way, but in this case, certain characteristics may be useful to observe on this small component for controlling the robot. Therefore, I chose the number of pixels containing the smallest size to be at least 5, but I did not rule out the possibility of using 15 pixels to see this size for very small components. So:

$$5 \leq a \leq 15 \quad (18)$$

From which, due to the 2.5 μm pixel size:

$$12,5 \mu m \leq a \cdot px \leq 37,5 \mu m \quad (19)$$

where:

a represents the number of pixels on which x can be seen,
 $a \cdot px$ is the size of x .

The magnification factor N can be calculated by using these:

$$N = \frac{a \cdot px}{x} \quad (20)$$

So the s' image distance can be calculated as:

$$s' = N \cdot |s| \quad (21)$$

Thus, the f focal length can be obtained by:

$$f = \frac{1}{\frac{1}{s'} - \frac{1}{s}} \quad (22)$$

Given the size range of the image of x and the range of possible object distances, there is also a range for the focal length. This is shown in Table 1, which demonstrates that it is quite wide.

Table 2: Range of focal lengths in millimeters for different object distances and image sizes ($f(a, s)$)

s [mm] \ a [-]	- 100	- 90	- 80	- 70	- 60	- 50	- 40	- 30	- 20
5	11.11	10.00	8.89	7.78	6.67	5.56	4.44	3.33	2.22
6	13.04	11.74	10.43	9.13	7.83	6.52	5.22	3.91	2.61
7	14.89	13.40	11.91	10.43	8.94	7.45	5.96	4.47	2.98
8	16.67	15.00	13.33	11.67	10.00	8.33	6.67	5.00	3.33
9	18.37	16.53	14.69	12.86	11.02	9.18	7.35	5.51	3.67
10	20.00	18.00	16.00	14.00	12.00	10.00	8.00	6.00	4.00
11	21.57	19.41	17.25	15.10	12.94	10.78	8.63	6.47	4.31
12	23.08	20.77	18.46	16.15	13.85	11.54	9.23	6.92	4.62
13	24.53	22.08	19.62	17.17	14.72	12.26	9.81	7.36	4.91
14	25.93	23.33	20.74	18.15	15.56	12.96	10.37	7.78	5.19
15	27.27	24.55	21.82	19.09	16.36	13.64	10.91	8.18	5.45

4.1.2 Selection of the camera system

Chapter 4.1.1 summarized the 0th step of the hardware selection process for the visual system. Following this, the iteration for selecting the camera system begins. Since specific products must be chosen, typically, discrete values or ranges of values are given for each parameter. That is, none of the parameters can be selected from a continuous set of values according to arbitrary requirements. Therefore, neither a lens that precisely satisfies the range of focal lengths in Table 2 nor a sensor with exactly the pixel size specified in Chapter 4.1 that is sufficiently high-resolution for observing the entire current device could be found. Thus, I examined several possible sensors and lenses from both technical and economic perspectives. Here, I present the system that was ultimately selected.

It is primarily worth examining the lower rows of Table 2, especially the left-hand columns. The reason for this is that it would obviously be good to see the examined size on as many pixels as possible, and as mentioned in section 4.1, very small object distances are less relevant. For this, the Waveshare Electronics 8-50mm Zoom Lens optics [27] may be a good choice for the approximately 8-27 mm focal length range. As the name of the product suggests, its focal length can be adjusted between 8 and 50 mm, and it is capable of a 3-megapixel resolution. With this focal length range, the object distance and magnification can be inversely calculated, and by knowing the pixel size, it can be verified whether the image of the smallest object size is on enough pixels. However, there are two problems with this. Firstly, the pixel size is not a fixed value, it depends on the chosen sensor. Secondly, the lens also has a spatial extent, which forces the sensor to a more distant point from the observed object. The first problem can be solved by comparing different sensors, and the second one can be solved by examining the value of the parameter as a function of the distance from the front of the lens and the focal length, rather than as a function of the object distance and focal length. To start with, based on equation (22):

$$s' = \frac{s \cdot f}{s + f} \quad (23)$$

Furthermore, based on equation (21):

$$N = \frac{s'}{|s|} \quad (24)$$

And according to equation (20):

$$a = \frac{N \cdot x}{px} \quad (25)$$

Combining equations (23), (24), and (25), we get:

$$a = \frac{x}{px} \cdot \frac{1}{|s|} \cdot \frac{1}{\frac{1}{f} + \frac{1}{s}} \quad (26)$$

It is essential to address the size of the optics here. Figure 14 can assist in this regard.

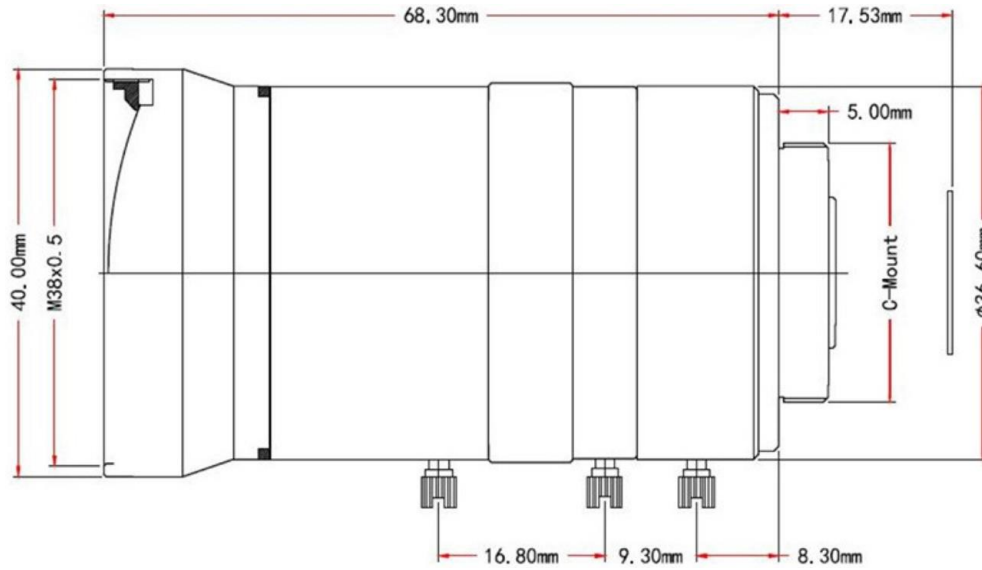


Figure 14: Waveshare Electronics 8-50mm Zoom Lens [27]

Let d be the total length based on Figure 14:

$$d = 68,30 \text{ mm} + 17,53 \text{ mm} = 85,83 \text{ mm} \quad (27)$$

The nodal point can be obtained by measuring the focal length along the optical axis from the sensor, and from here the object distance can be measured [26]. This is because the object and image space of the optical system have the same refractive index, so the nodal points and principal points coincide. Thus, the size of the

object distance can be calculated based on the distance from the lens, the total length of the lens, and the focal length.

$$|s| = |s_o| + d - f \quad (28)$$

where, not previously marked:

s_o is the distance of the object from the lens.

By considering the sign:

$$s = s_o - d + f \quad (29)$$

And this can be substituted into (26), so:

$$a = \frac{x}{px} \cdot \frac{1}{|s_o - d + f|} \cdot \frac{1}{\frac{1}{f} + \frac{1}{s_o - d + f}} \quad (30)$$

So now only the selection of the appropriate sensor needs to be solved. One suitable choice is the Sony IMX477 with a $1.55 \mu\text{m}$ vertical and horizontal pixel size and a resolution of 12.3 megapixels [28]. This results in a diagonal pixel size of $2.2 \mu\text{m}$ and a higher resolution than the optics, so it does not degrade the design in this respect. Examination of the $a(s_o, f)$ with these data can be seen in Figure 15.

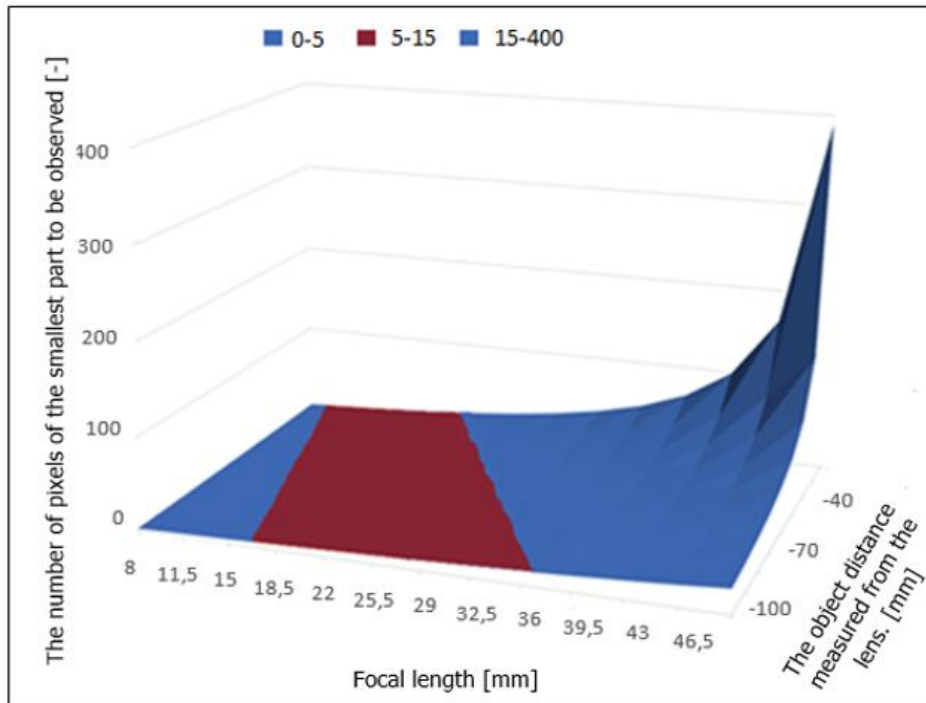


Figure 15: Minimum number of pixels containing a given size as a function of various distances and focal lengths

The result of observing with too few pixels would be inaccurate robot navigation, while representing too many pixels would result in the entire observed device not fitting into the image, potentially providing

insufficient information. Figure 15 shows that for any distance, there is a suitable focal length setting that produces the appropriate number of pixels. The favourable range of these settings is shown in the red-coloured area. Thus, I have successfully implemented a flexible, multifunctional solution that robustly satisfies many demands as mentioned in section 4.1. This flexibility can also be observed in Figure 16, which shows the range of distances that can be achieved with different focal length and minimum pixel size settings.

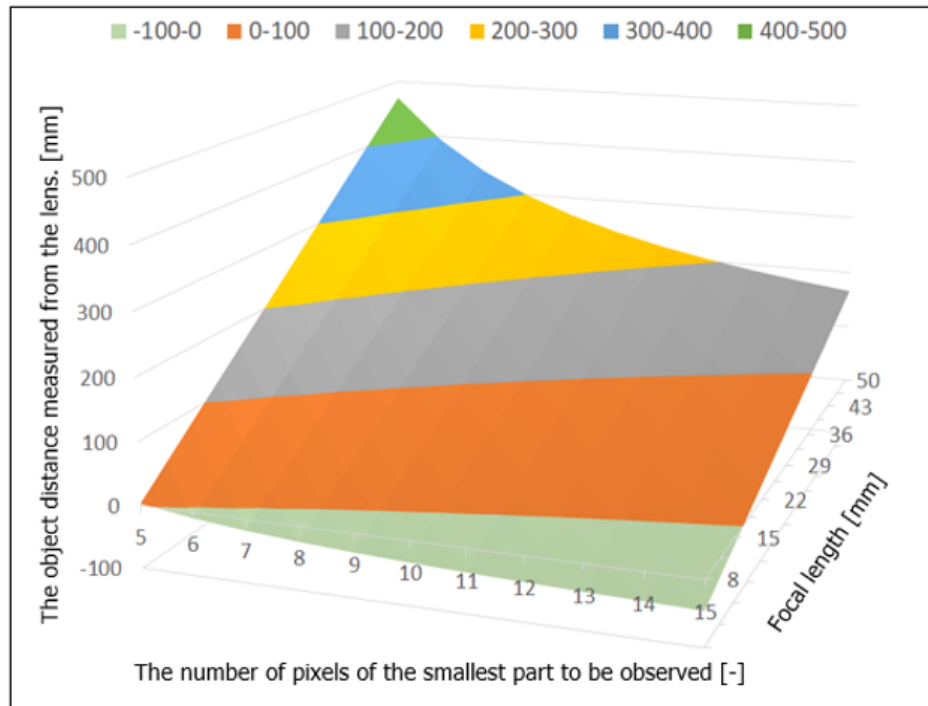


Figure 16: Available distances from the lens for different focal lengths and smallest pixel sizes

In Figure 16, the negative range is purely theoretical, as the object to be observed cannot be placed inside the lens. The relevant part is the range of 20-100 mm measured from the lens, where it can be seen that all defined minimum pixel sizes can be achieved with the system, making the solution fully satisfying all requirements. In addition, it is important to note that when selecting such a camera system, the sensor format is a crucial parameter. The sensor format and the optics must match each other. If the optics' sensor format is too large, the image it creates will protrude beyond the sensor. If it is too small, the edges of the image will remain dark and only a circular area in the centre will contain useful information. The sensor format of the Sony IMX477 and the Waveshare Electronics 8-50mm Zoom Lens are the same, making them compatible with each other. This optical system forms the basis of the work. Of course, it can be technically replaced with other alternatives [29-32], but considering economic factors as well, the one presented here is the most suitable.

4.2 Hardware resources and mechanical design

With this optical system, everything is in place. The selected sensor is available in the Raspberry Pi HQ Camera [33], so the Raspberry Pi provides an excellent opportunity to create a standalone camera platform.

I had the Raspberry Pi 4 Model B [34] hardware at my disposal, which I connected to the camera with its own factory ribbon cable. This 2000 mm cable is long enough to allow the Raspberry Pi to stay stationary while the camera is attached to the robot. Therefore, I designed a camera holder for the system using AutoCAD Inventor Professional 2022 software.

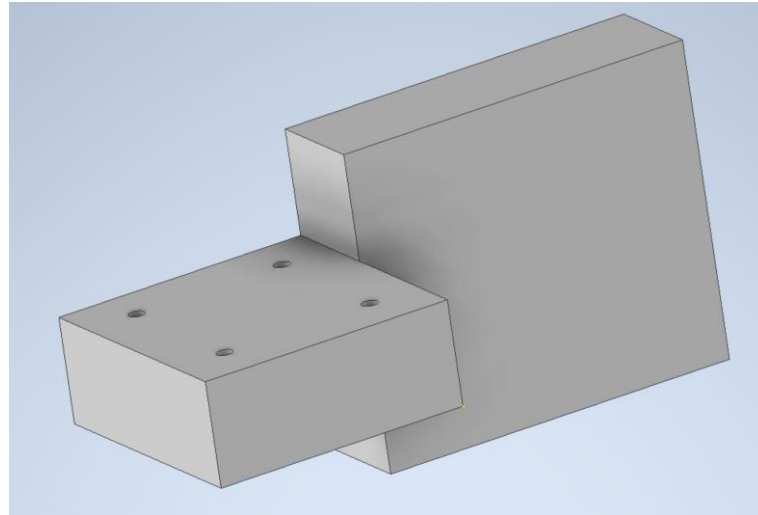


Figure 17: Camera holder in Autocad Inventor Professional 2022 software

Figure 17 illustrates this support device that aids in the mechanical implementation. Figure 18 depicts the dimensions of the device, which were designed to match the robot's specifications. The top view dimensions of the left block in Figure 17 can be seen on the left side, while the side view dimensions of the right block are shown on the right side. In the first case, the body of the left block was extended by 20 millimetres, and in the second case, the body of the right block was extended by 70 millimetres. I printed the component with the Creality CR-X Pro 3D printer.

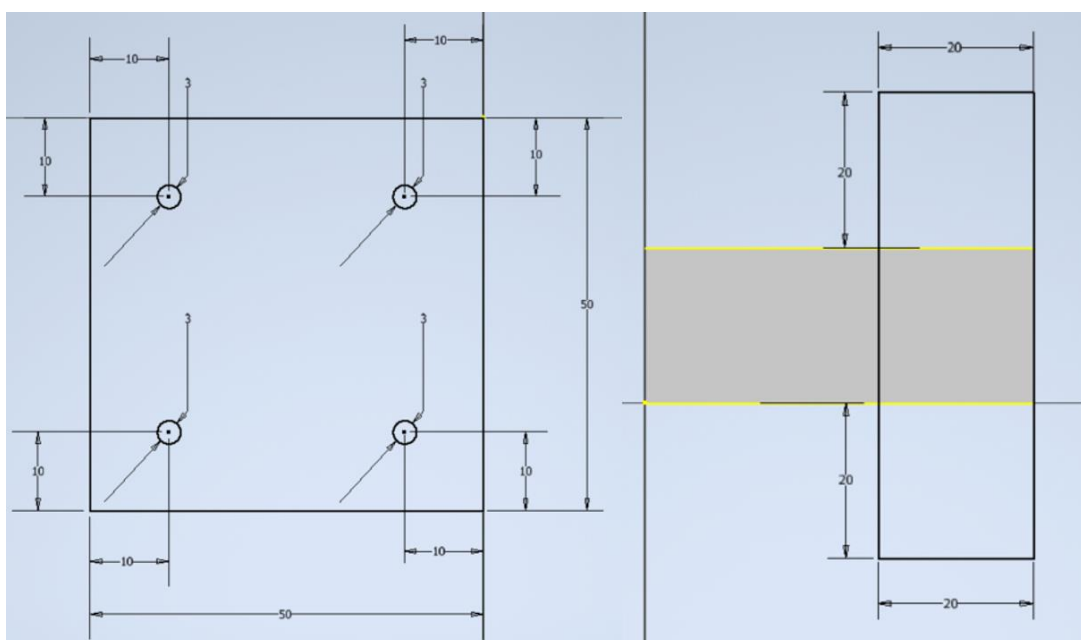


Figure 18: Dimensions of the camera holder

5 Software implementation

In this chapter, I provide an overview of the software I developed for my system, and detail and discuss the scientific achievements, implementation of individual autonomous subsystems, the codes and algorithms applied and implemented, as well as the literature used.

5.1 Transmitting camera feed from Raspberry Pi server to client machines

On the server side of my system, a script is continuously running on the Raspberry Pi 4 Model B platform mentioned in Chapter 4.2. I create a socket and allow multiple sockets to connect to the same port. Then, I specify the IP address and port number, and wait for the connection. Once it is established, I capture the current photo with the camera and send it. On the client side, if I want to receive an image in any part of the code, I create a socket, connect to the server using its IP address and port number, and receive the file in packets and save it. These image files are used for the configuration steps and also for taking photos during measurements.

5.2 Configuration of the vision system

As mentioned in section 3.1.3, automated measurement is preceded by configuration performed by a human. This subsection presents the configuration of the software subsystem, which is basically divided into three parts:

- Calibration of the distortion of the optical system.
- Calibration of the position and orientation of the optical system.
- Placement of measurement samples and setting the position of the robot for contacting the first sample.

After this, the measurement can start according to section 3.1.3 and Figure 13, for which my system provides the necessary input data. The software implementation of the first two points listed here is presented in sections 5.2.1 and 5.2.2.

5.2.1 Calibration of the distortion of the optical system

In some cases, calibration of distortion can be considered as part of the configuration, since if, for example, the size of the measured object changes, or the object needs to be observed from a different distance, it may be necessary to adjust the focal length. However, if objects with the same size are measured consecutively, the focal length remains fixed and the previous calibration is adequate, since it corresponds to the same state of the optical system.

5.2.1.1 Equations used for distortion calibration

Radial distortion occurs because the optical system does not perfectly converge non-parallel incoming light rays, especially those arriving at the edge of the system's lens, resulting in the image appearing at a different pixel location on the sensor. According to the work of Drap and Lefèvre [35], distortion should be calculated using equation (31):

$$\begin{pmatrix} x_d \\ y_d \end{pmatrix} = \begin{pmatrix} x_u \\ y_u \end{pmatrix} + \begin{pmatrix} x_u \\ y_u \end{pmatrix} \cdot \sum_i k_i \cdot r^{2i} \quad (31)$$

where:

- $\begin{pmatrix} x_d \\ y_d \end{pmatrix}$ represents the distorted pixel position in terms of x and y coordinates,
- $\begin{pmatrix} x_u \\ y_u \end{pmatrix}$ represents the correct pixel position in terms of x and y coordinates,
- r is the distance from the centre,
- k_i represents the i-th constant coefficient describing the distortion.

At this point, the projection matrix is not yet known, but according to [36], an algorithm can still be initiated. If the relative positions of certain points in the image are known and their world coordinate system location is also known, then by rearranging equation (31), we get:

$$\begin{pmatrix} x_d \\ y_d \end{pmatrix} - \begin{pmatrix} x_u \\ y_u \end{pmatrix} = \begin{pmatrix} x_u \\ y_u \end{pmatrix} \cdot \sum_i k_i \cdot r^{2i} \quad (32)$$

In this case, the left-hand side of equation (32) represents the distortion in pixels. By minimizing the magnitude on the right-hand side for different positions, the set of distortion coefficients can be obtained for which the distorted pixel positions have the smallest least squares error. *OpenCV* uses the Levenberg–Marquardt algorithm [37] for this nonlinear minimization, and avoids the trivial solution where $k_i = 0$ for all i. It should be noted that, as mentioned earlier, the camera matrix is not yet known, but it is necessary for projecting real-world points since equations (31) and (32) can be applied to these projected points. Therefore, *OpenCV* calculates the matrix using the widely known SVD decomposition [38].

5.2.1.2 Implementation of distortion calibration

According to section 5.2.1.1, the known points with relative positions are the corners of a chessboard, which is shown in a calibration image in Figure 19. Similar photos need to be taken with the camera and run through the code, which uses *OpenCV* and other libraries such as *numpy*.



Figure 19: Distortion calibration using a chessboard pattern

According to Figure 19, the world coordinate system is fixed in the chessboard plane along the z-axis, and as shown, only the internal corner points are considered, of which there are $9 \cdot 10 = 90$ points. Along the orange, red, pink, blue, turquoise, green, lemon, orange, red path drawn on the figure, the points $(0, 0, 0)$, $(1, 0, 0)$, ... $(9, 8, 0)$ can be obtained in the z-plane, respectively. These points are then used to create a $9 \cdot 10$ array using *numpy*. Following this, separate lists are created for the real points and image points, where every point from every image can be stored, and the list of images is read, and iteration through the images begins. Each image is converted to grayscale, and the *OpenCV findChessboardCorners* function [39] is used to find the corners of the current chessboard. This function takes in the image, the dimensions of the chessboard, and can also be given various flags. Among these, it may be useful to set adaptive thresholding so that the algorithm is not sensitive to lighting, and normalization for histogram equalization. The corners found in this way can be refined using the *cornerSubPix* function [40]. The world and image points are then added to the created lists, and the iteration moves on to the next image. At the end of the iteration, the *calibrateCamera* function is called, giving these lists. This works according to section 5.2.1.1. This way, the projection matrix and the constant coefficients describing the distortion can be obtained, and these are saved. It is important to note that when creating calibration images, care should be taken to ensure that a sufficient quantity and quality of images are taken for the sake of numerical stability. An important characteristic of quality is that the chessboards should be as close to the edges of the images as possible, as this is where the radial distortion is the greatest. Furthermore, it should be noted that although the functions discussed here are called from Python, they actually run in a C++ environment for the sake of runtime efficiency.

5.2.2 Calibration of the position and orientation of the optical system [41]

The calibration of the position and orientation of the optical system is fully part of the above mentioned configuration. This must be done before every measurement is started, as it would not be easy to set the relative position of the camera and the measurement space to $100 \mu\text{m}$ accuracy before every measurement. This can be attempted, but a truly good and robust solution should not assume such accuracy of human steps. As a result of any potential changes in position and orientation, the same real object may appear differently in the image. The individual shapes are projected onto the image through perspective projection. Perspective projection is not distance-preserving, not scale-preserving, not angle-preserving, not parallelism-preserving,

but it is collinear, meaning that it preserves straight lines. It is actually a description of the pinhole camera model, as the projecting lines pass through a single point. Like many other projection, this can also be characterized using a matrix.:

$$\begin{pmatrix} s \cdot x' \\ s \cdot y' \\ s \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & b_1 \\ a_{21} & a_{22} & b_2 \\ c_1 & c_2 & 1 \end{pmatrix} \cdot \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \quad (33)$$

where:

s is the scaling factor,

x' is the horizontal coordinate in the projected image,

y' is the vertical coordinate in the projected image,

$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$ is the matrix describing the orientation part of the transformation,

$\begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$ is the vector describing the translational part of the transformation,

$(c_1 \ c_2)$ is the projection vector,

x is the original horizontal coordinate in the object space,

y is the original vertical coordinate in the object space.

The mapping matrix can be calculated based on the knowledge of 4 original-projected point pairs, since the matrix is defined by 8 constants. This matrix is important for the application because it defines a quadrilateral region on the projected image, which is then transformed into a rectangular region. Based on the information contained in the rectangular region, the robot can navigate through the environment. In practice, a printed sheet containing markers is placed in the measurement area, and an image is taken of it. This is illustrated in Figure 20.

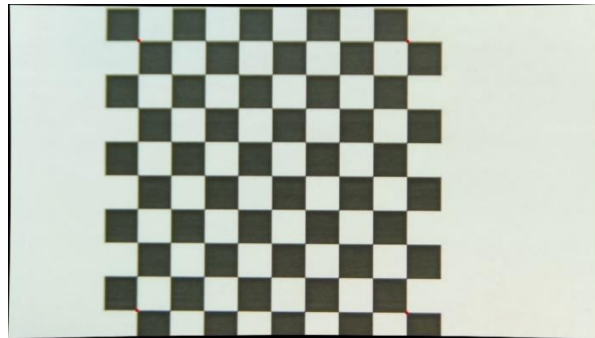


Figure 20: Image of position and orientation calibration

It can be seen that distortion calibration has already been performed here, and the outer curves of the image have been pulled inward by the algorithm described in section 5.2.1. The chessboard on this image is designed in such a way that the rectangle defined by the four inner red corners precisely identifies one measurement location where DBC can be determined. In the case of the configuration before the measurement, the robot performs this calibration at this one location, but the measurement algorithm described in section 3.1.3 applies the obtained transformation at every time.

So, the complete code can be characterized as follows. I apply distortion calibration using the *OpenCV* functions *getOptimalNewCameraMatrix* and *undistort*. Important input parameters are the projection matrix calculated according to 5.2.1 and the distortion coefficients. Then, I locate the four points marked in red on Figure 20 using the appropriate indexing and the *findChessboardCorners* function. I refine the corners again using the *cornerSubPix* function, then I obtain the perspective projection matrix using *getPerspectiveTransform* according to (33), which I save. Later, when reading in individual images, the *warpPerspective* function can perform the necessary transformation.

5.3 Displacement determination

The appropriate quality images are now available. Therefore, in each case, it is necessary to determine the required correction by comparing two images, as discussed in section 3.1.3 with reference to Figure 13. My idea is to first perform semantic segmentation to determine the domain where the measured DBC is located in the already good quality, undistorted images. Then, I search for characteristic points on these images that belong together. From these, I select the reliable points and determine the actual displacement based on their average displacement. Also, I determine the average rotation based on the average rotation of the vectors they form. I return these values as return values to the robot. Based on this, the entire software module can be modularly illustrated as follows:

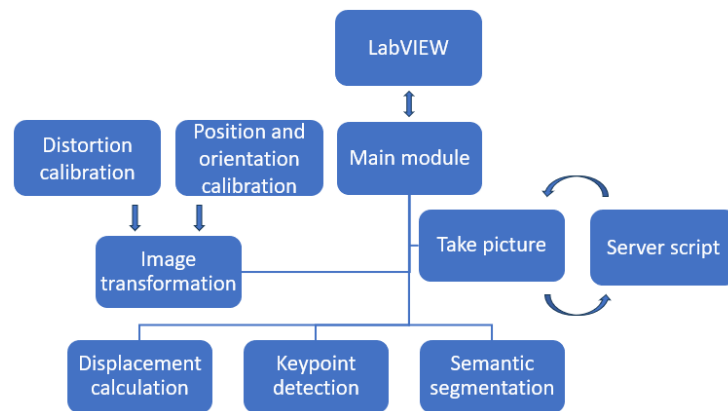


Figure 21: Modular structure of software

The main module communicates with LabVIEW software. Using the calibration scripts discussed in sections 5.2.1 and 5.2.2, a transformation can be performed on the image sent from the server mentioned in section 5.1 to produce a good quality, essential region from the image. From this region, the *DBC can be segmented*, and *key points can be searched*, and *the displacement can be determined* relative to the previous image. In the following sections, I will discuss these three processes.

5.4 Semantic segmentation

The essence of semantic segmentation is to determine the class of each pixel in an image. Thus, its output resolution is the same as the input resolution, and for each pixel, there can be as many output values as the

number of classes to choose from for the given task. In my case, I need to determine for each pixel whether it belongs to the DBC or not, resulting in two possible values, 1 and 0. The IoU (Intersection over Union) value is a widely used and accepted quality metric for this decision, quantifying the overlap between the segmentation output and the real segmentation. Its description is as follows:

$$IoU = \frac{\text{output of segmentation} \cap \text{real segmentation}}{\text{output of segmentation} \cup \text{real segmentation}} \quad (34)$$

Typically, during evaluation, the IoU value is calculated for each class and averaged. Another possible quality metric is pixel accuracy. Here, the number of correctly classified pixels is compared to the total number of pixels:

$$\text{Pixel accuracy} = \frac{\text{correctly classified pixels}}{\text{total number of pixels}} \quad (35)$$

Using this is useful, for example when running a machine learning algorithm, but alone it is not really suitable for evaluating segmentation. The reason for this is that in cases where one class appears in very small proportions in an image, its substantial non-recognition does not cause significant errors in pixel accuracy. Therefore, inadequate semantic segmentations may seem appropriate if only this characterization is used. This is also true for the average class accuracy metric:

$$\text{Average class accuracy} = \frac{1}{\text{number of classes}} \sum_{\text{class}} \frac{\text{correctly classified pixels}_{\text{class}}}{\text{total number of pixels}_{\text{class}}} \quad (36)$$

The essence of this metric is to sum the ratio of correctly classified pixels to the total number of pixels for each class and divide the total by the number of classes. In my work, I used artificial intelligence-based neural networks for semantic segmentation. I reviewed a lot of research papers, publications, books, and source code to find the best possible method. In the next chapter, I will summarize my research work concisely.

5.4.1 Literature summary

Semantic segmentation is not coincidentally difficult to implement well, as it is one of the greatest challenges in computer vision. As mentioned in Chapter 5.4, its task is to make pixel-by-pixel decisions on a given image about which class a particular pixel belongs to. It is possible to carry out semantic segmentation by conventional methods [42-50], but as in many other areas of science, deep learning methods play a significant role here. There are numerous applications of semantic segmentation implemented with deep learning. Medical applications are presented in [51]-[55], while self-driving cars utilize it in [56]-[59]. Notable applications also include semantic segmentation related to wildlife and animals [60]-[62], and pedestrian segmentation [63][64], which also promotes autonomous transportation. Furthermore, semantic segmentation is used in various other fields, such as defect detection during material inspection [65], or satellite image analysis [66][67]. Different authors classify the techniques used for various applications in

different ways. Guo et al. [68] provide a comprehensive overview of deep learning-based semantic segmentation and divide the literature on different techniques into three main parts: region-based, fully convolutional network-based, and weakly supervised methods. However, Lateef and Ruichek [69] use a much more detailed classification, categorizing the methods into ten categories. These methods are based on techniques such as Markov fields and conditional probabilities, their replacements, resolution changes, recurrent neural networks, regions, deconvolution, part extraction, weak and medium reinforcement, temporality, and detail encoding. Li et al. [70] apply an even deeper division. They divide the previously mentioned methods into unimodal and multimodal considerations, essentially contrasting methods that use a single frame and those based on three-dimensional procedures composed of consecutive, temporally different images or simply multiple images. However, the classifications in [71]-[73] are not identical either. In terms of applications, the fully convolutional network (FCN) has performed very well among the techniques listed above. Diverse examples include road segmentation [74], segmentation of high-resolution aerial images [75], tree species segmentation [76], and segmentation performed on MRI scans [77]. It is no coincidence that the authors of [74]-[77] and many others [78]-[81] chose this method. From the semantic segmentation review articles mentioned earlier, it is clear that FCN is considered outstanding. According to Lateef and Ruichek [69], FCN brought the real breakthrough in deep learning-based semantic segmentation. Li et al. [70] also believe that following the techniques based on Markov fields and conditional probabilities, and methods using simple convolutional neural networks, truly good semantic segmentation solutions were brought by FCN. Similarly, according to the article by Yu et al. [71], recent successes in segmentation are primarily due to FCN. Hao et al. [72] claim that it drastically increased accuracy, while Garcia et al. [73] emphasize that the currently most successful state-of-the-art deep learning semantic segmentation techniques are all based on FCN. As summarized in [68]-[73], numerous FCN-based techniques have already been developed. These are solutions of various sizes and thus have different hardware requirements. Table 3 shows FCN methods and further techniques based on their considerations.

Table 3: FCN methods and related techniques

Data type Method	Benchmark	Average IoU [%]	Reference
SDN	PASCAL VOC 2012 test	79.9	[82]
OA-Seg	PASCAL VOC 2012 test	74.1	[83]
DeconvNet	PASCAL VOC 2012 test	69.6	[84]
FC-DenseNet103	CamVid dataset	66.9	[85]
SegNet	CamVid dataset	60.1	[86][86]
FCN-8s	PASCAL VOC 2011 segval	62.7	[87]
FCN-16s	PASCAL VOC 2011 segval	62.4	[87]
FCN-32s	PASCAL VOC 2011 segval	59.4	[87]

The method for calculating the average IoU values, which best characterize the accuracy mentioned here, is discussed in detail in the previous Chapter 5.4. The individual benchmarks are internationally accepted, well-known datasets, which are specifically suitable for verifying semantic segmentation [88]-[90].

5.4.1.1 *Articles related to semantic segmentation of electronic devices*

Although it is evident that deep learning-based semantic segmentation is popular and widely used, the topic of electronic devices, particularly in the case of components mounted on DBC, is published to a very limited extent and can be considered modern and worth scientific investigation. I did not find any journal or conference articles specifically on DBC semantic segmentation. Fault detection methods can be found for IGBTs, which are often placed on DBC [91], or for printed circuit boards [92], but these require different techniques. Other literature performed instance segmentation for capacitors and resistors [93]. However, the success of this method would be of doubtful value for my application since every component in the published training and testing dataset was in front of a single-colored, white background, and only their shadows could be considered as distractions. Thus, the method would not be robust enough for my application. Interesting publications on printed circuit boards dealing with components can also be found, but these focus on detection [94] or depth-based segmentation of larger elements [95]. None of the electronics-related articles I found were published before 2020, so it can be confidently stated that the topic is novel and open for discussion.

5.4.2 Implementation

To create a technically appropriate, fully convolutional neural network suitable for semantic segmentation, I implemented my own code. The dataset needed for training was prepared using the same camera and optics as in the case of live measurements. I performed the annotation manually on the training and validation images. I modified the optimization algorithm, dilation, batch size, channel number, depth, epoch count, regularization weighting, learning rate, learning rate decay ratio, and decay step size, thus training and evaluating neural networks 12,149 times with the help of automated runs. In the following, I will characterize separately the related code snippets, classes, functions, and the overall operation of the network that I have prepared for this purpose. To do this, I used the Python Pytorch [96], Torchvision [97], Pillow [98], Numpy [99], Matplotlib [100], OS [101], and Glob [102] libraries. I note that I will use lowercase names for these libraries in the future, staying true to the format of my code.

5.4.2.1 *Data augmentation function*

Just as in other areas of artificial intelligence training, the appropriate amount of data is necessary for robust learning. Therefore, I implemented my own data augmentation. To do this, I created a class and overrode the `()`, or `__call__`, operator. If the `__call__` method is called and an image is given with the help of *pillow*, a

random number is generated uniformly between 0 and 1. If the number is greater than 0.5, the image is reflected along its horizontal axis; otherwise, it is not. The overridden function returns the image.

5.4.2.2 Dataset preparation

The software implementation of the data set was done as follows. I derived a custom class from the *pytorch torch.utils.data.DataLoader* class. I overrode its *init* method to take in arguments specifying which data augmentation to apply to the images and segmented images in the current image or segmented image and where the root folder of the images and segmented images are located. From the former, I create attributes for the class, while for the latter, I use the *os.path.join* and *glob.glob1* functions to create a list of paths to the images and segmented images, which I then sort. I also create attributes from these lists.

Next, I make the class indexable, so I override the *getitem* method. I set it up so that indexing the attribute of the list of image paths returns the corresponding image, and I do the same for the segmented images. Using the *open* function from the *pillow* library's *Image* module, I then save a reference to the current image and its label in image and label variables, respectively. Finally, I call the *random.seed* function twice with the same, randomly generated seed variable and, if data augmentation was specified for the image or segmented image, apply it after the calls and return the references.

5.4.2.3 Creation of convolutional layers

In my network, a convolutional layer includes not only the convolution operation but also nonlinearity and normalization. I derive this as a class from the *pytorch.nn* module. I also override the constructor function, which takes in the input and output channel numbers, the convolution kernel size, and the *stride* parameter that specifies how many pixels the kernel should move through the image. In addition, I allow for *dilation* to be added to the convolutional layer. This parameter determines whether the $k \cdot k$ convolution kernel processes adjacent pixels or applies a single pixel skip, as illustrated in Figure 22. Essentially, this option can increase the convolution kernel's field of view but sacrifices some of the pixels near its centre. Whether or not to use this option depends on the application, and I test this parameter along with others in my work.

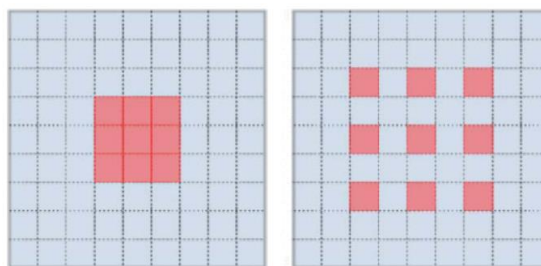


Figure 22: Convolutional kernel without dilation and with dilation [103]

After initializing the parent class, I create a *pytorch.nn.Conv2d* object as the *convolution* attribute. This is called with the channel numbers, *stride*, and *padding*, which specifies how many 0s to add to the edges of

the image. I calculate the padding by dividing the kernel size by 2, adding 1 to the quotient if there is dilation. I also create an attribute for normalization, using the `pytorch.nn.BatchNorm2d` function, which takes in the output channel number.

To move from one layer to the next, i.e., from one object to the next, I use the *forward* function. This function returns the normalization with an improved linear function, the Rectified Linear Unit (ReLU), by substituting the first created *convolution* attribute. *ReLU* is an internationally accepted and widely used method [104][105].

5.4.2.4 Creating a transposed convolutional layer

I create this class in a formally very similar way to the convolutional layer. I use the same parent class and the overridden constructor function also receives the same inputs. However, the *convolution* attribute here is an object of the `pytorch.nn.Transpose2d` class. It also receives the input and output channel numbers, as well as the *stride* and *padding* parameters. The difference from my convolutional layer is that, due to the property of transposed convolution, the pixels generated at the edges of the image must be discarded, which must be indicated in the form of a parameter. The *forward* function is the same as the *forward* function of my convolutional layer.

5.4.2.5 Creating fully convolutional architectures

With this, I have all the building blocks at my disposal to create a fully convolutional architecture. As I mentioned in the introduction to section 5.4.2, I trained networks 12149 times, starting from as many different network states. One of the variable parameters was the depth of the network, that is, how many convolutional and transposed convolutional layers I used. So, in this regard, I created six different architectures. It is important to highlight this as it is a branching point in the object initialization.

In each case, I also derive my class from the `pytorch.nn` module, where during initialization, I need to provide the input channel number, depth, and dilation. As a first step, I create a convolutional layer as an attribute, which takes the 3 colour channels as input and returns the architecture's input channel number. Then n downsampling and convolution steps follow each other, which are also implemented using convolutional layers. The parameters of the i -th downsampling layer are: the architecture's input channel number 2^{i-1} times, twice of this, the kernel size, the *stride*, and the *dilation*. In the case of convolution, both parameters are 2^i times the architecture's input channel number. After this, I create an attribute from n transposed convolutional layers, where the parameters of the i -th layer are 2^{n-i+1} times and 2^{n-i} times the architecture's input channel number. Finally, I create a classifier, which is an instance of the `pytorch.nn` module's *Conv2d* class. Its parameters are the output channel number of the last transposed convolutional layer of the architecture, the number of classes, which in my case is 2, as well as the kernel size and the associated *padding*. I fixed these latter two values to 5 and 2, respectively, so that the last layer can see a larger area.

In the *forward* function written here, we first need to store the references of the results of the $n + 1$ convolutional layers in local variables. Then we need to add these to the outputs of the corresponding transposed convolutional layers. Finally, we need to return the return value of the last modified transposed convolutional layer of the classifier. This addition, that is, connecting the upscaling and downscaling parts, is a recognized and reliable method to help the convergence of the training [106][107].

5.4.2.6 Creating training and validation functions

The inputs of my *trainer* function are the current epoch number, a reference to the network, an object instance belonging to my custom dataset class, an optimization algorithm from the *pytorch.optim* library with an initial gradient of zero, a criterion class from the *pytorch.nn* module, and the width and height of the currently used images. To measure accuracy, I create three local variables: one that counts how many images the algorithm has used, one that shows the average pixel accuracy among them, and one that shows the error for the current epoch. Following this, it is necessary to set the network to training mode using the *train* function of the *pytorch.nn.Module* class, and then iterate through my input dataset with a loop, since I have overridden its `__getitem__` function and thus can access individual images and segmented images. I also request the index of these using *enumerate*. I would like to note that the hardware I used for training has an NVIDIA Quadro RTX 3000 graphics card and version number 452.57, so I was able to use the CUDA 11.0.3 Update 1 driver for faster performance [108]. Inside the loop, I call the built network and pass its output as input to the criterion class invoked with the `__call__` operator. This returns a *loss* value, for which the *backward* function performs derivation on each parameter. With this calculation, the optimizer's parameters can be updated using the *step* function. After updating, I deal with the statistics, increasing the error for the current epoch with the scalar value stored in the *loss*, and for each pixel of each image, I get the most likely estimated class from the network output using *pytorch.max*. I modify the used images variable and the accuracy variable, then log the current epoch number, the *loss* value, and the accuracy, and finally return the latter two.

The *validation* function differs only slightly from this. In addition to the previously mentioned image counter, accuracy determining, and average accuracy per epoch variables, I also calculate and log the IoU and average class accuracy values according to Section 5.4.

5.4.2.7 Creating a run function

Now the training can begin, for which I use my own *run* function. This function sets up the depth of the network, input channel size, batch size, optimizer algorithm, epoch number, regularization weighting, learning rate, learning rate decay rate, and dilation step size for learning, or without it. Firstly, for reproducibility, I fix the internal state of all used tools with the appropriate seed value, which I set to 42. These tools include *random*, *numpy.random*, *pytorch*, and *pytorch.cuda*.

The assembly of the training, validation, and testing datasets is done with my dedicated function. Here, in addition to my custom data augmentation used only in training, I define further transformations. For both images and segmented images, I perform resizing, using bilinear interpolation in the former case and nearest-neighbor interpolation in the latter. Then, I create the training database with the given batch size. Finally, before starting the training loop, I define the criterion class, optimization algorithm, and method for modifying the step size using the decay rate and step size. My criterion is the widely used cross-entropy [109][110].

5.4.2.8 Main function

Finally, I implemented the main function, with which I create several fully convolutional architectures within the ranges specified in Table 4 for the adjustable parameters.

Table 4: Ranges of hyperparameters to be optimized

Hyperparameter	Minimum value	Maximum value
Depth of network	1	7
Channel number	6	16
Learning rate	10^{-4}	10^{-2}
Learning rate decay rate	0.1	0.5
Step size	10	30
Regularization weighting	10^{-5}	10^{-3}
Batch size	1	16
Dilation	0	1
Epoch number	40	90

The table does not include it, but I chose the Adam algorithm [111][112] and the Stochastic Gradient Method [113] as the optimizer algorithms, as the former is currently considered one of the best in international practice, and the latter can be useful if my training is not complex enough for the dataset. The former method proved to be more successful. The ratio of training to validation images was 337-85. I manually performed the necessary annotations using makesense.ai [114], as mentioned in the introduction to section 5.4.2. The entire data preparation, network building, and execution are well illustrated in Figure 23.

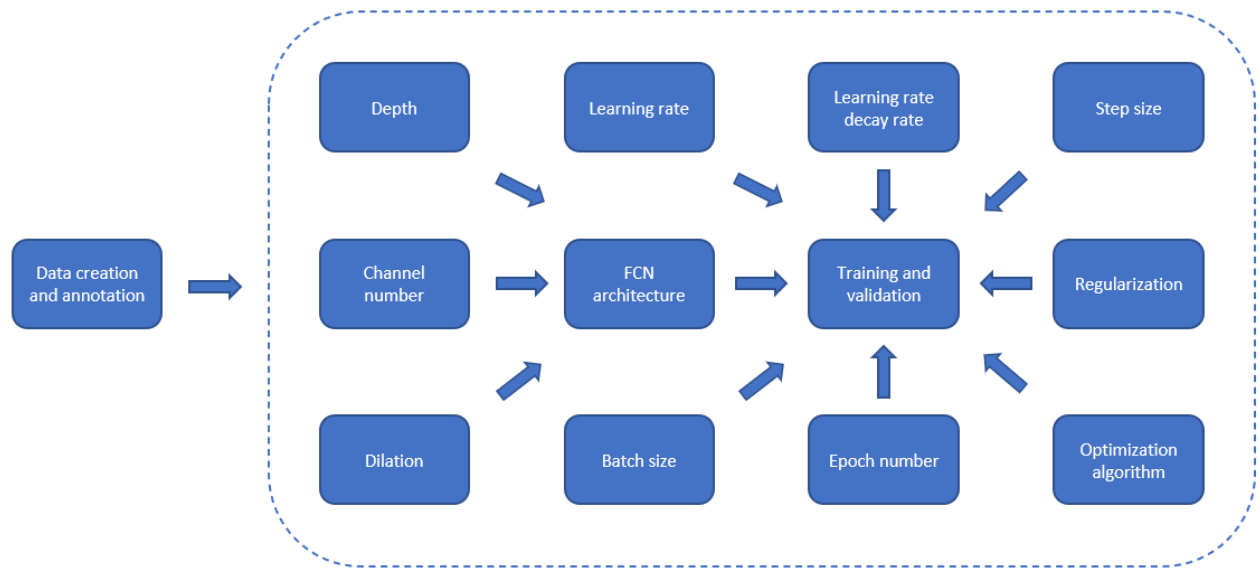


Figure 23: Process of training the FCN architecture

However, we have not yet discussed how I optimize the hyperparameters of the system, including the optimizer algorithm, dilation, batch size, number of channels, depth, number of epochs, regularization weight, learning rate, as well as the learning rate decay rate and decay step size, within the ranges mentioned in Table 4. This is covered in section 5.4.2.9.

5.4.2.9 Hyperparameter optimization

Hyperparameter optimization is one of the most crucial aspects of developing learning algorithms [115]. If I want to find the best solution to a data-driven problem, after selecting the problem-solving model, I also need to fine-tune its settings and hyperparameters to obtain the most accurate solution. The use of deep neural networks in this problem is a good choice because they are proven to be suitable for learning complex non-linear relationships, such as those involved in semantic segmentation. To optimize the hyperparameters of these convolutional neural networks, I need to find a good methodology. It is important to find a good methodology because evaluating the training for different settings of convolutional neural networks, especially those working with a large amount of data, can be a time-consuming process. Therefore, a good approach is needed to obtain the best solution as quickly as possible. Thus, Figure 23 can be complemented as follows:

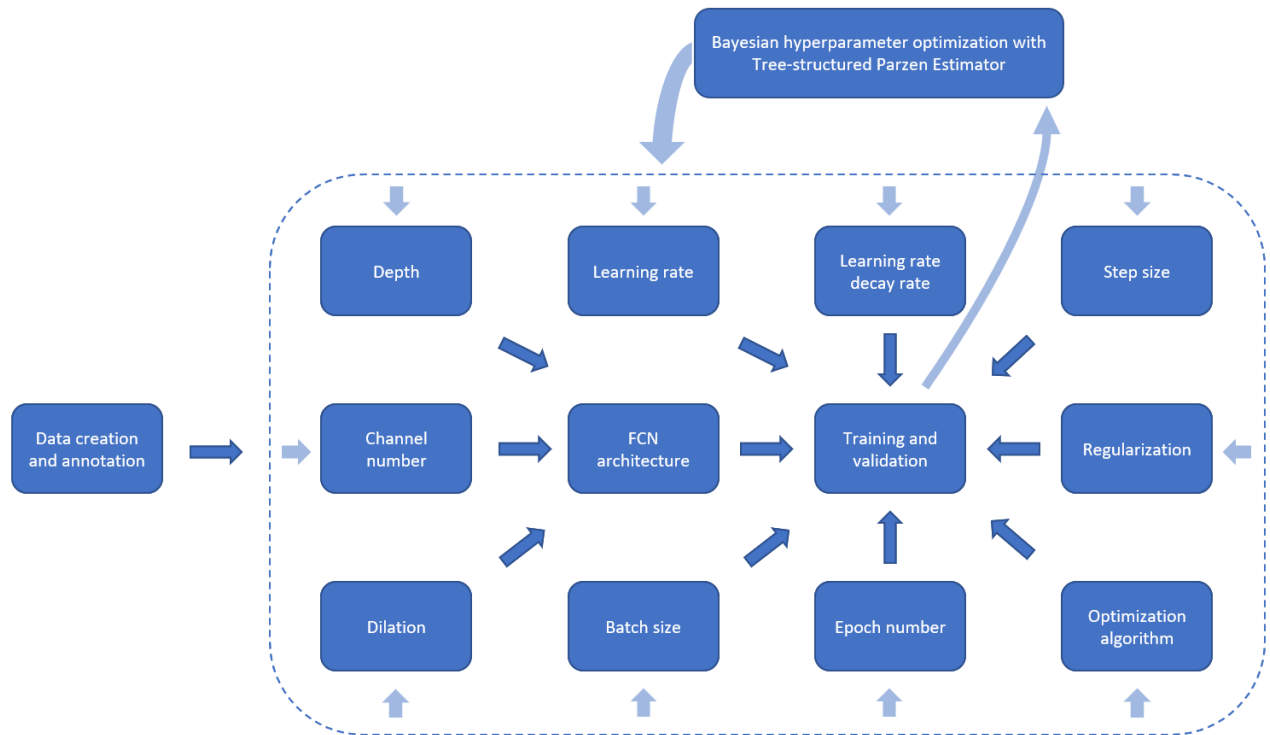


Figure 24: Process of hyperparameter optimization for the FCN architecture

Manual, grid, random, and algorithmic optimization

There are several options for optimizing hyperparameters [116]. The simplest one from a software development perspective is the manual method. In this case, each value is manually modified, and new runs are started, while observing which settings yield better results from the learning algorithm. Another solution that requires slightly more attention is when all possible model configurations within a given range are explored using multiple nested cycles. This method is more advantageous in the sense that it does not require continuous human presence, but it is still extremely time-consuming. Therefore, if the speed of optimization is also a concern, it can only find the local optimum of a smaller neighborhood of the hyperparameter space. Random search is also an option, but there is a risk of missing the best hyperparameter configurations. However, it can be faster than grid search.

The most efficient and modern method is algorithmic optimization, as it can intelligently exploit the results of previous iterations within a single run. In general, most modern hyperparameter optimization algorithms are based on Bayesian optimization [117], although there are cases where gradient-based optimization [118] or evolutionary algorithms [119] can be applied, depending on the specific system being optimized and its evaluability. The essence of Bayesian hyperparameter optimization is that, at each optimization iteration, it builds a probability model of the objective function to be optimized, i.e., the error function of the neural network in the hyperparameter space, and uses this model to determine what hyperparameter configuration to evaluate the actual objective function at the next step. The main difference between algorithms that

implement Bayesian optimization, or Sequential Model-Based Optimization (SMBO), is how they construct the probability model of the objective function. Of the two most popular algorithms, Gaussian Processes (GP) and Tree-structured Parzen Estimator (TPE), I chose the latter because it was more effective in several tests [115]. It has been used, for example, in optimizing solar power forecasting [120], fault diagnosis of rotating machinery [121], evaluating landslide susceptibility [122], and health management applications [123], among many others [124-127].

Bayesian hyperparameter optimization with TPE

Bayesian hyperparameter optimization can be described by the following flowchart:

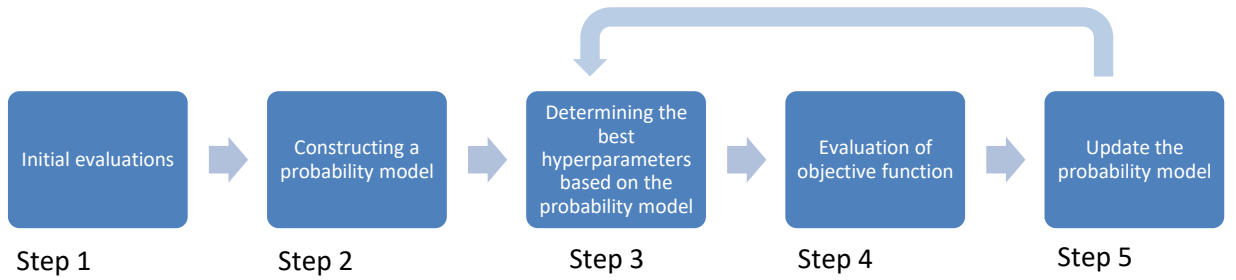


Figure 25: Bayesian optimization

It can be seen that the first and second steps only need to be performed once, while the third, fourth, and fifth steps need to be repeated cyclically. The stopping criterion is typically based on time, number of iterations, or the value of the objective function. The first step involves the initial random exploration of the simulation space to provide a basis for building the probabilistic model. In my case, I used the best initial configuration determined through experimental trial and error using *Optuna* [128], along with 10 random iteration configurations. The second and fifth steps involve TPE-specific modelling, the third step involves decision-making based on the TPE-specific criterion, and the fourth step involves running the neural network.

It is worth starting with the criterion. What makes a hyperparameter configuration good? To determine this, TPE uses the Expected Improvement (EI) function:

$$EI_{y^*}(x) = \int_{-\infty}^{y^*} (y^* - y) \cdot p(y|x) dy \quad (37)$$

where:

- x is the current hyperparameter configuration,
- y^* is the threshold value of the objective function. According to the best initial configuration determined through experimental trial and error using *Optuna*, this is the lower decile. After the 25th iteration, it is the smaller value between the lower decile and the 25th smallest value,
- y is the current value of the objective function,
- $p(y|x)$ is the conditional probability model of the objective function.

It is already clear from this formula that the algorithm searches for objective function values that are smaller than the threshold value, since this is the only way the integral value can be positive. Therefore, the goal is to minimize the error. Understanding the concept of expected improvement, it is worth moving on to the main component of TPE, the probability model:

$$EI_{y^*}(x) = \int_{-\infty}^{y^*} (y^* - y) \cdot \frac{p(x|y) \cdot p(y)}{p(x)} dy \quad (38)$$

where:

$p(x|y)$ is the conditional probability model of the hyperparameter configuration,

$p(y)$ is the probability model of the objective function,

$p(x)$ is the probability model of the hyperparameter configuration.

Furthermore, TPE introduces the functions $l(x)$ and $g(x)$ such that:

$$p(x|y) = \begin{cases} l(x) & | y < y^* \\ g(x) & | y \geq y^* \end{cases} \quad (39)$$

Here, the $l(x)$ and $g(x)$ models are constructed as Gaussian mixture models (GMM) [129], where for each data point, a distribution is fit with a variance equal to the distance to the closest data points on both sides.

Thus, if we extract $\frac{1}{p(x)}$ from (38), we get:

$$EI_{y^*}(x) = \frac{1}{p(x)} \int_{-\infty}^{y^*} (y^* - y) \cdot p(x|y) \cdot p(y) dy \quad (40)$$

Then, based on the Expected Improvement formula (39), it can be written as:

$$EI_{y^*}(x) = \frac{1}{p(y < y^*) \cdot l(x) + (1 - p(y < y^*)) \cdot g(x)} \int_{-\infty}^{y^*} (y^* - y) \cdot l(x) \cdot p(y) dy \quad (41)$$

From this, we can calculate:

$$EI_{y^*}(x) = \frac{p(y < y^*) \cdot y^* - \int_{-\infty}^{y^*} p(y) dy}{p(y < y^*) + (1 - p(y < y^*)) \cdot \frac{g(x)}{l(x)}} \quad (42)$$

This shows that if the expression $\frac{g(x)}{l(x)}$ decreases, then the denominator of the fraction in (42) also decreases, and thus the fraction, i.e., $EI_{y^*}(x)$ increases. This means that there is a higher chance of improving the

estimate. Essentially, this means that TPE algorithm always tries to choose a hyperparameter configuration that is close to the configurations that result in smaller errors, by looking at the ratio of objective function values above and below the threshold.

5.4.2.10 Evaluation of segmentation

As introduced in section 5.4, the internationally accepted measure for evaluating the quality of semantic segmentation is Intersection over Union. In Table 3, I demonstrated the results achieved with various well-known methods on widely used benchmarks. In my case, I was looking for a solution to a specific task. For semantic segmentation, the goal was to accurately identify the pixels of DBC tools. This problem can be considered less complex than, for example, segmenting different types of vehicles and animals in the PASCAL VOC 2012 dataset [88], which is why better results can be expected. I tried to build smaller, more compact neural networks than those listed in Table 3. The reason for this was that a complex network developed for 20 object categories is probably not the most suitable for my specific problem. My results supported this expectation.

As described in section 5.4.2.9, the Parzen Estimator selects hyperparameters in the simulation space where state estimation performs the best. In my case, the performance was measured by the strictest IoU metric, which can comprehensively evaluate the results of an estimate as described in the introduction to section 5.4. The result of the optimization is summarized in Figure 26.

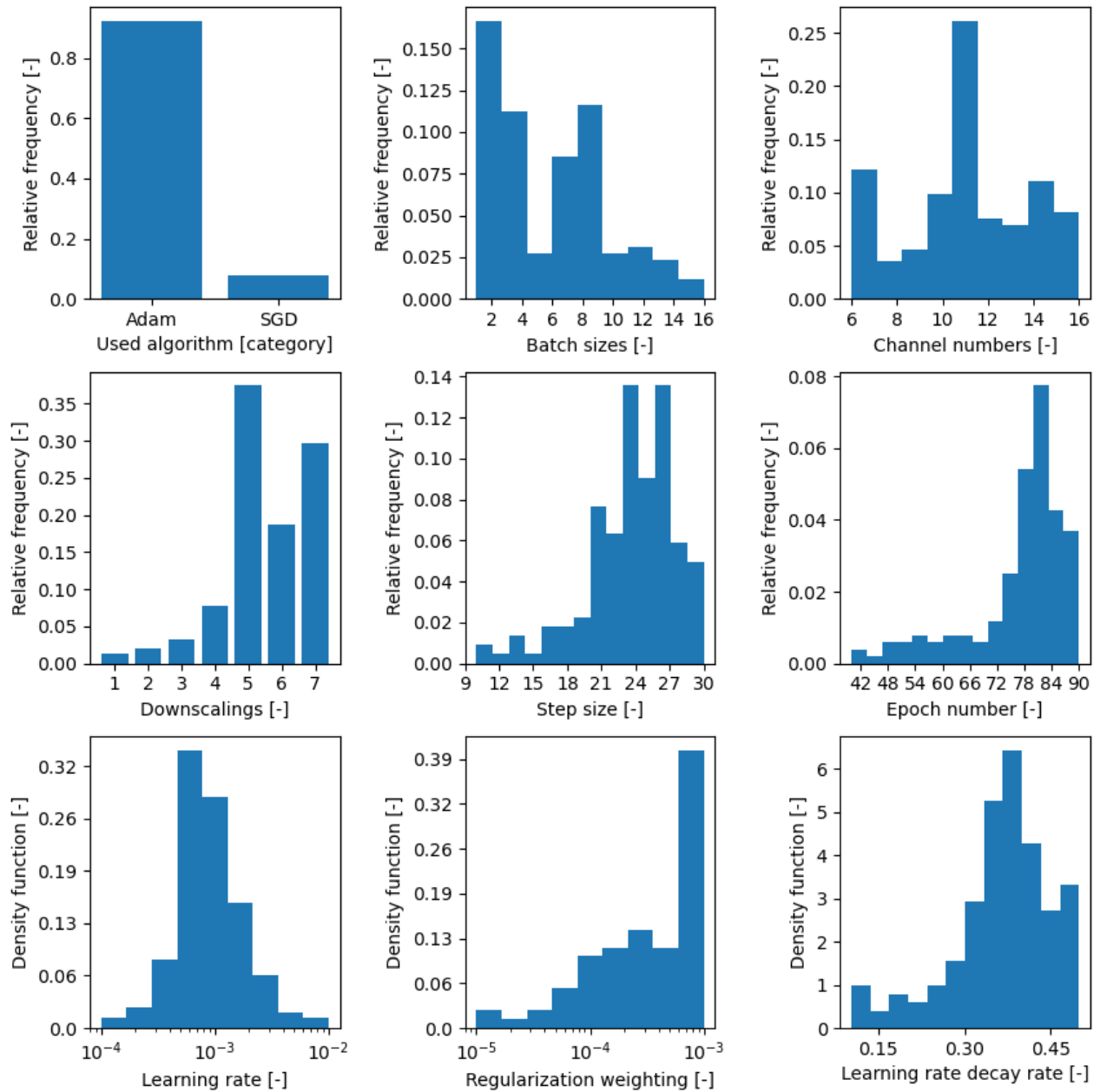


Figure 26: Relative frequency results of hyperparameter optimization

The Adam algorithm has clearly proven to be better in terms of optimization. This is not so surprising, since semantic segmentation is a complex problem and the training dataset created for it is small in proportion, so it is not necessary to use the SGD method as described in section 5.4.2.8.

Among the batch sizes, the smaller ones proved to be suitable, so the training works better if the weight parameter optimization of the convolutional kernels is done based on only a few images at a time. The optimal number of channels was around 11, meaning that the system works best with this many parallel kernels. As for the scaling, the inner point of the optimization interval, specifically 5, proved to be the best. Therefore, it is not good to use too many scaling operations, as important information for semantic segmentation can be lost, but if the network depth is too small, sufficiently complex image features cannot

be extracted. The optimal step size was in the range of 21-27. If it is smaller, the learning rate will decrease too quickly, and if it is larger, the weight parameter optimization will not be fine-grained enough. Among the epoch numbers, a large number around 80 was optimal, indicating that overfitting should not be a concern, even in long training sessions, as mentioned in [130]. The learning rate takes on a value that is roughly around 0.001, as in the literature [111], so this parameter is not unique. This rate is multiplied by a value of roughly 0.35 for the chosen step size, which is the ideal decay rate. Regularization should be taken into account for larger weights, as this effectively avoids overfitting. The dilation values were not represented, and their distribution was 50-50%, so the segmentation result is independent of the field of view of individual layers. Therefore, it is better to work with the case without dilation for computational time reduction.

The solution that performs the best, with an IoU value of 98.37%, uses the Adam algorithm with a batch size of 2, 11 channels, a depth of 5, a step size of 26, 80 epochs, a learning rate of 0.000583, a regularization weight of 0.000987, and a decay rate of 0.37, without dilation. The learning process of this solution is illustrated in Figures 27-29.

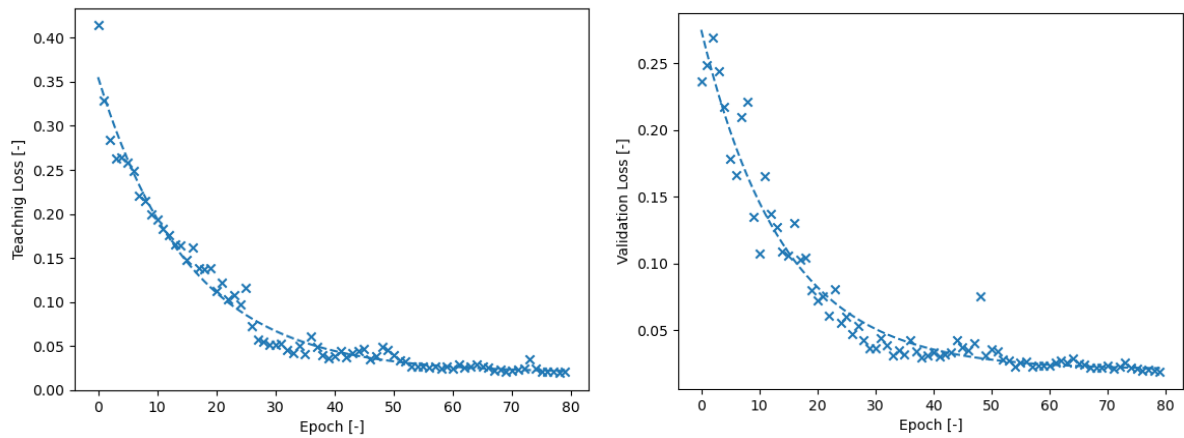


Figure 27: Losses of the best run

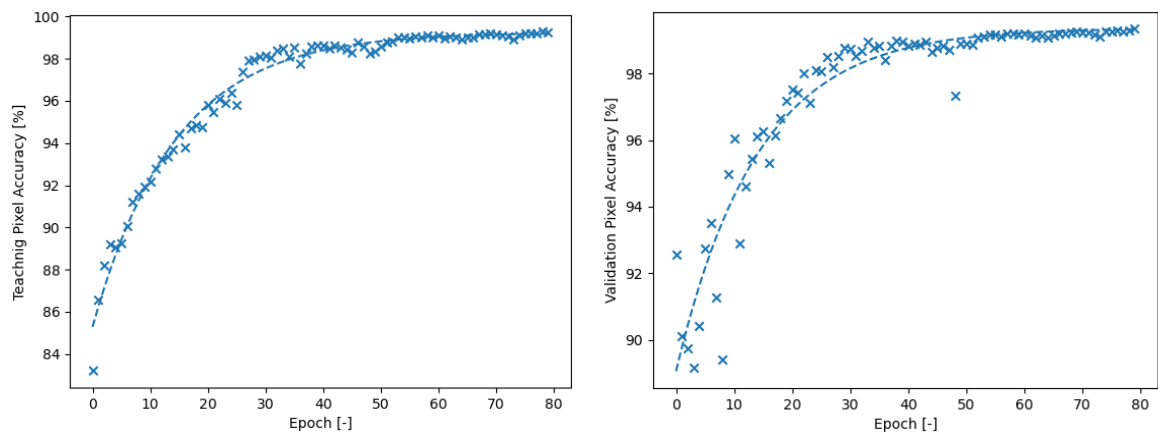


Figure 28: Pixel accuracies of the best run

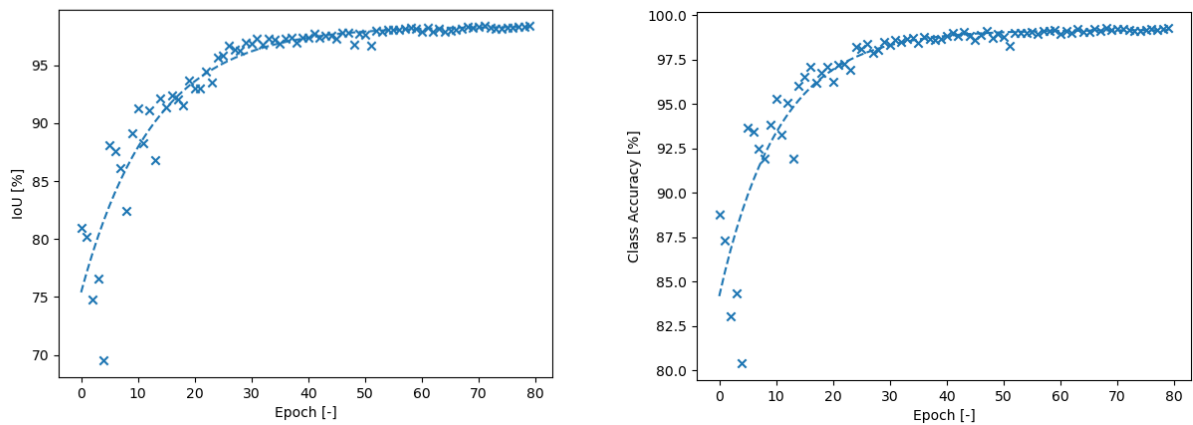


Figure 29: Intersection over Union and class accuracy values of the best run

Thanks to the Tree-structured Parzen Estimator, the well-chosen architecture and methodology show a nice, uniform convergence. The validation results are very similar to the training results in terms of both loss and pixel accuracy values, which rules out overfitting. The obtained IoU value of 98.37% is outstanding compared to the results presented in Table 3 at the beginning of the chapter.



Figure 30: Result of semantic segmentation

Additional visual representation, along with a complete evaluation of the architecture, can be found in section 5.7. The efficiency of hyperparameter optimization is also well demonstrated by the approximate density functions obtained from the results returned by the training runs. It is apparent that using a poorly constructed set of hyperparameters can lead to 10-20% worse results in terms of pixel accuracy, class accuracy, and IoU metric. Therefore, with the help of a modern optimization technique, I was able to find a configuration that produces really good results from a relatively wide range of values. The wide range of results is illustrated by Figure 31, which shows the training losses, validation losses, training pixel accuracies, validation pixel accuracies, class accuracies, and IoU values.

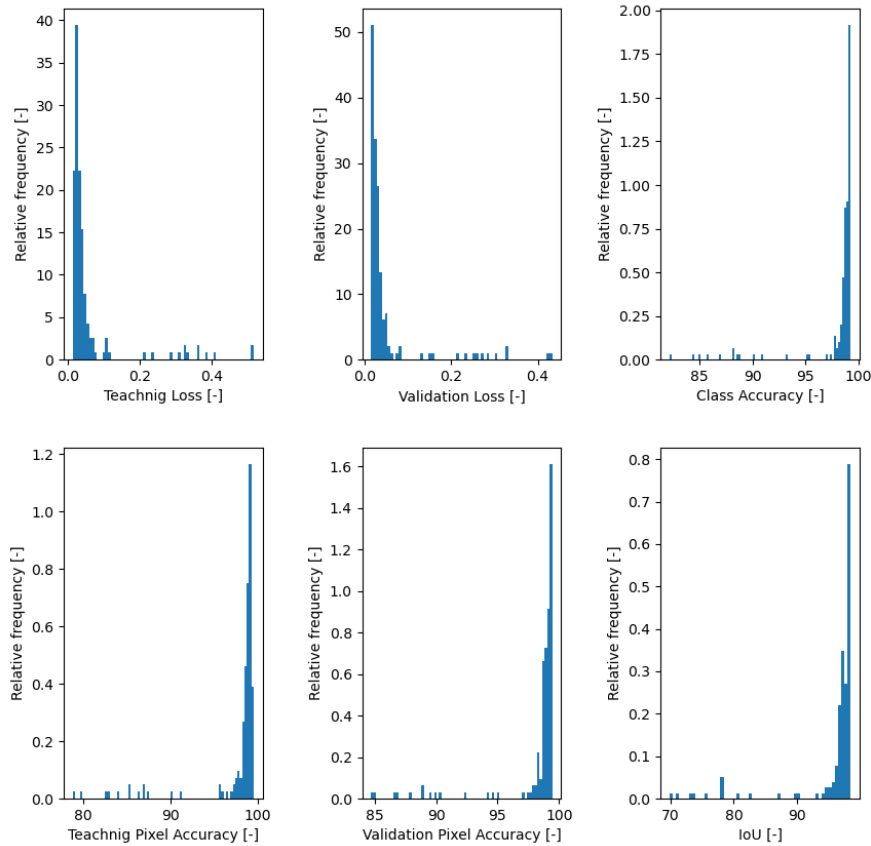


Figure 31: Wide range of results of the optimization

5.5 Keypoint detection [131][132]

In chapter 5.4, I determined the range in which the appropriate keypoints can be searched for the calculation of displacement. I found these keypoints using the SIFT (Scale Invariant Feature Transform) detector, whose patent expired in March 2020, making it now free to use in industry.

This algorithm essentially looks for corner points and creates a descriptor that is invariant to all transformations except for perspective distortion based on their local neighbourhoods. To achieve this, it uses the DoG (Difference of Gaussians) filter, which is widely used in edge detection problems. The algorithm runs the filter in multiple sizes on the image and searches for local maxima according to three parameters: the filter size, and the x and y coordinates. These local maxima are not discrete in nature but rather continuous since SIFT fits a multi-variable polynomial to the calculated points. As a result, it achieves a *higher level of accuracy than pixel-level*, which is particularly important in my application. The generated descriptor is 128-dimensional and is stored in a one-dimensional array. It is created from various directional intensity gradients within a 16x16 pixel window around each detected point, with weighting. The code takes into account the derivatives farther from the corner point less than the closer ones. The direction of the largest derivative is the reference direction from which the descriptor's combined intensity gradient can be rotated. For example, if I run it on an image and its rotated version, it still finds the same keypoints in both cases.

Among the 36 derivatives obtained every 10 degrees, if there is one whose value reaches 80% of the original corner point, a descriptor is also created for it. Additionally, the algorithm divides this already rotated 16x16 neighbourhood into 16 4x4 neighbourhoods, normalizes them to 1, and performs the process again with 45-degree resolution instead of 10, thus ensuring scale invariance. This gives the final descriptor. Therefore, normalizing to 1 ensures insensitivity to linear changes in brightness. If the change is nonlinear, it does not significantly reduce the efficiency of the method, as the normalization is actually done by setting a threshold value (0.2) for each gradient, and modifying values below it before normalization. This makes the orientation of the individual descriptors more prominent than their magnitude.

5.6 Displacement calculation

So, if the reference image and the image to be analyzed have already undergone semantic segmentation and the SIFT detector has been used to find their keypoints and descriptors, then the correction calculation can be performed.

5.6.1 Shift correction

First, I will present the shift correction. To do this, I first pair the individual points based on their descriptors using the match function of the OpenCV *BFMatcher* class [133]. Based on this, I can obtain the confidence-ordered pairs of points. The user can set in advance in the configuration file how many of these point pairs the algorithm should use in the future. Typically, the optimal value falls in the range of 40-60, and every option was satisfactory for all my tests. I will discuss this in more detail in section 5.7.

Next, I calculate the differences between the x and y coordinates of these point pairs and determine their average and standard deviation using the *numpy.mean* and *numpy.stdev* functions. Then, I exclude the point pairs corresponding to differences outside the standard deviation. This way, out of the originally selected 40-60 high-quality detections, only the 30-40 best remain. It is necessary to perform two different quality filters in two steps because this is the only way to filter out errors, such as when the SIFT detector detects two different legs of a D2PAK as identical. Such errors are typically rare and result in a significantly different distance than other values, so they are sure to disappear by excluding the standard deviation. Finally, I calculate the average of the remaining differences, obtaining the appropriate shift correction.

5.6.2 Rotation correction

Of the two corrections, rotation correction is the more difficult to calculate. Essentially, when selecting the value of *n* mentioned in chapter 5.6.1, the user can only choose even values. The reason for this is that the best point pairs selected in this way are paired for rotation calculation and vectors are defined between them. Vectors are generated in both the reference image and the image under examination, and the rotation value can be obtained from the average of their rotations. This is mathematically justified. Essentially, the

DBC in the image is a planar shape with its centre of gravity initially located at the origin, around which rotation occurs, followed by a translation of the entire DBC. These transformations result in the following changes at an arbitrary i-th point in the DBC:

$$\begin{pmatrix} 1 & 0 & a \\ 0 & 1 & b \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos(\alpha) & -\sin(\alpha) & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} x_i \\ y_i \\ 1 \end{pmatrix} = \begin{pmatrix} x_i \cdot \cos(\alpha) - y_i \cdot \sin(\alpha) + a \\ x_i \cdot \sin(\alpha) + y_i \cdot \cos(\alpha) + b \\ 1 \end{pmatrix} \quad (43)$$

where:

- x_i is the x-coordinate of the i-th point,
- y_i is the y-coordinate of the i-th point,
- α is the angle of rotation,
- a is the x-component of translation,
- b is the y-component of translation.

Furthermore, I take a vector pointing from an arbitrary j-th point to an arbitrary k-th point in the DBC, and based on the rotated and translated points obtained from this vector's points based on (43):

$$\overrightarrow{JK} = \begin{pmatrix} x_k - x_j \\ y_k - y_j \\ 0 \end{pmatrix} \quad (44)$$

$$\overrightarrow{J'K'} = \begin{pmatrix} x_k \cdot \cos(\alpha) - y_k \cdot \sin(\alpha) + a \\ x_k \cdot \sin(\alpha) + y_k \cdot \cos(\alpha) + b \\ 1 \end{pmatrix} - \begin{pmatrix} x_j \cdot \cos(\alpha) - y_j \cdot \sin(\alpha) + a \\ x_j \cdot \sin(\alpha) + y_j \cdot \cos(\alpha) + b \\ 1 \end{pmatrix} \quad (45)$$

And from these:

$$\overrightarrow{J'K'} = \begin{pmatrix} (x_k - x_j) \cdot \cos(\alpha) - (y_k - y_j) \cdot \sin(\alpha) \\ (x_k - x_j) \cdot \sin(\alpha) + (y_k - y_j) \cdot \cos(\alpha) \\ 0 \end{pmatrix} \quad (46)$$

And this is equivalent to rotating the vector pointing from point J to point K by an angle of α around the origin. Therefore, the angles between the vectors indicate the rotation of DBC.

So, after determining the position vectors defined by each pair of points, you calculate their angle with respect to the x-axis using the *mat.atan* function. By subtracting these angles from each other, I obtain the angle of inclination of each vector. Similar to section 5.6.1, I calculate the mean and standard deviation of these angles, and then only consider the results within the range of the standard deviation to calculate the final result.

5.7 Evaluation of results

With this, I have reached the evaluation of my software's results. In terms of semantic segmentation, I have already performed a quality characterization related to my work in Chapter 5.4.2.10. Now, I will compare this with previously referenced literature and, subsequently, examine whether the entire system meets the 250 μm displacement and 1° rotational accuracy requirements.

5.7.1 International aspects of semantic segmentation

In general, for semantic segmentation, the IoU values of individual FCN networks range from 60 to 80% when examining various major benchmarks, as discussed in the literature review, Chapter 5.4.1 [82]-[90]. Of course, better results can be achieved for more specialized tasks with different architectures. As I mentioned, the field of semantic segmentation of electronic devices and DBCs is very poorly published in my case, but special applications can be examined from other areas. For example, in the introduction of [51], several urban applications were collected with IoU values of 80 to 90%. Among medical applications, I would highlight [54], where they achieved an average IoU value of 93.21% in the case of colorectal polyp segmentation. In the highly researched and highly important road segmentation for the development of future self-driving cars, [56] listed values between 85.9% and 96.7%. In terms of computer vision, in a lower quality snowy environment, this number is around 50% according to [57]. Defects on metal surfaces were determined by [65] using several methods with values between 80 and 90%, and [67] segmented individual city quarters based on satellite images with up to 87.43%. Thus, the performance of my novel application, which is over 98%, absolutely holds its own among special cases and does not perform worse than them.

5.7.2 Examination of accuracy

A reliable and robust operation was an important requirement for my entire system, both for me and my team. Therefore, I conducted a detailed analysis by creating real measurement conditions. I set up my camera and ran my code, which took a picture of a DBC. This was the DBC that corresponded to the reference according to Figure 13. Then, I placed another DBC, belonging to the same sample set, in a different location and ran my code on it as well. It worked as shown in Figure 32, providing the correction values, the displacement vector, and the rotation.

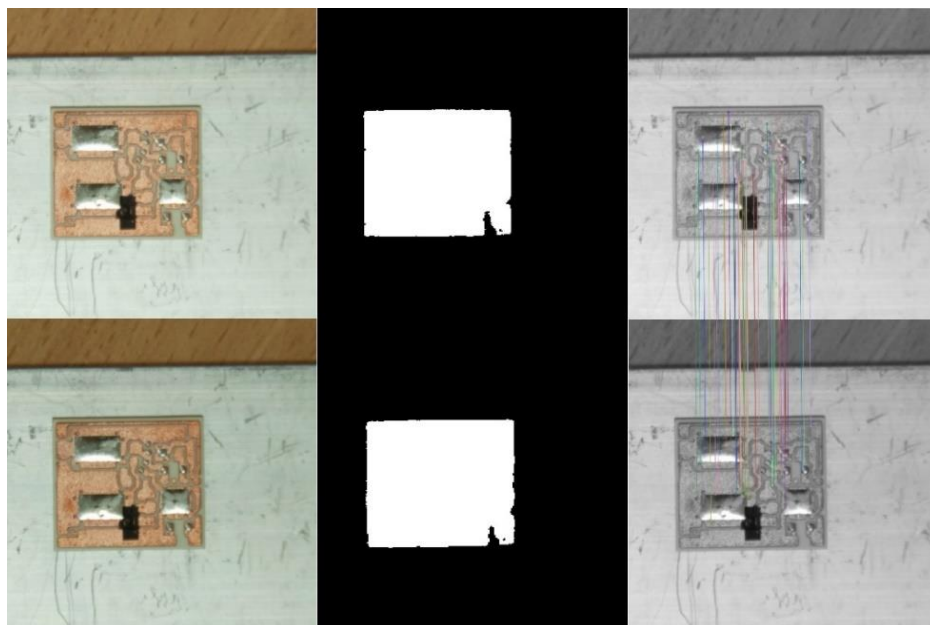


Figure 32: Results of the semantic segmentation and the execution of the SIFT algorithm

I verified these by measuring the displacement of individual points with a micrometer with a $10\text{ }\mu\text{m}$ accuracy and calculated the rotation from these vectors as well. Considering the corners of each DBC's bounding rectangle, I determined 4 points, which I treated as keypoints and performed the measurement algorithm using the micrometer. I did this in 50 cases, where I received an appropriate estimate each time. I will show three of these in detail here, marking the keypoints found by the software and those examined with the micrometer. I combined this examination with the investigation of how many points, according to Chapter 5.6.1, the user should specify as appropriate. In Figure 32, the reference DBCs can be seen in the top row, while the DBCs to be examined are in the bottom row. Figures 33 to 41 summarize the results of the correction calculation, which is a quality characteristic of the entire system.

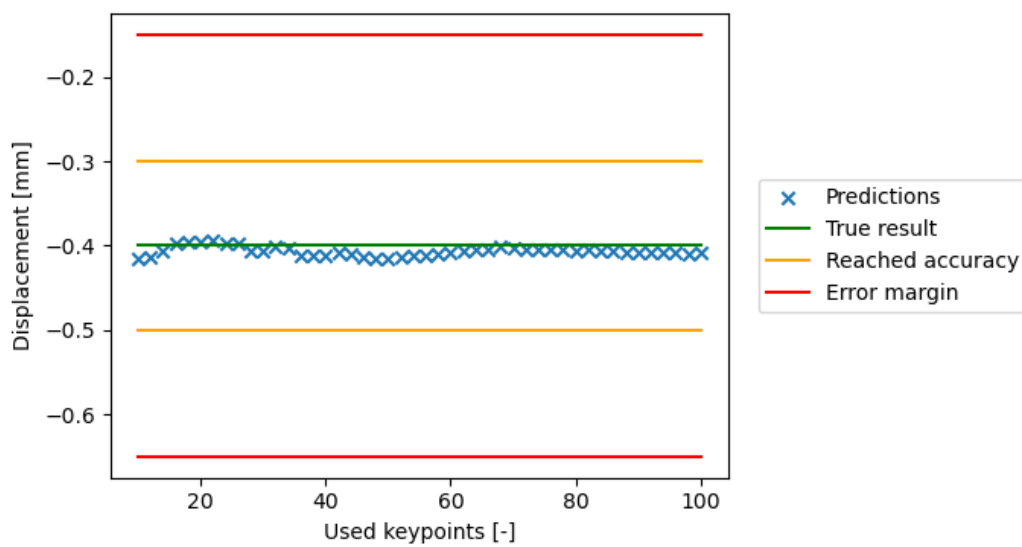


Figure 33: Displacement calculated along the X-axis 1.

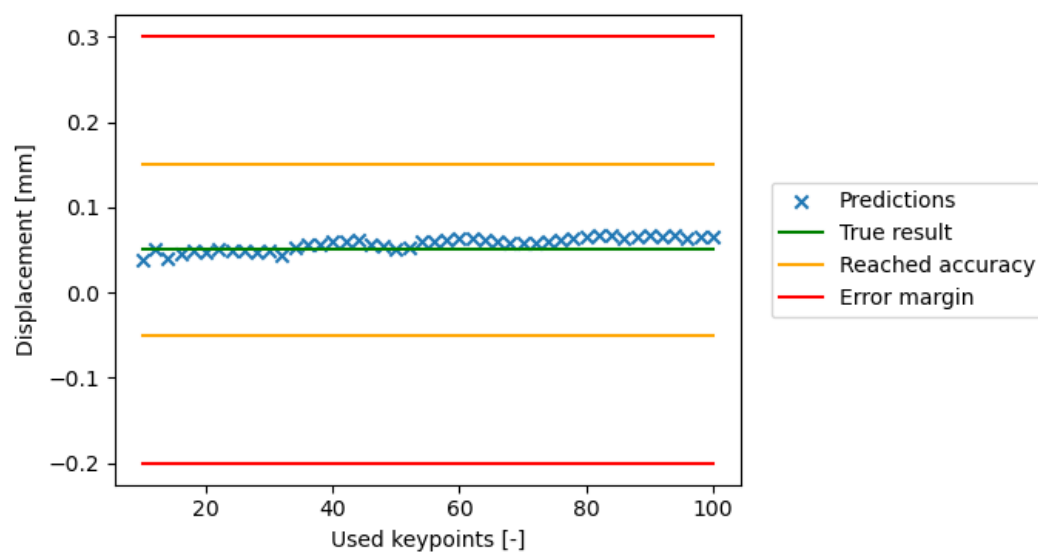


Figure 34: Displacement calculated along the Y-axis 1.

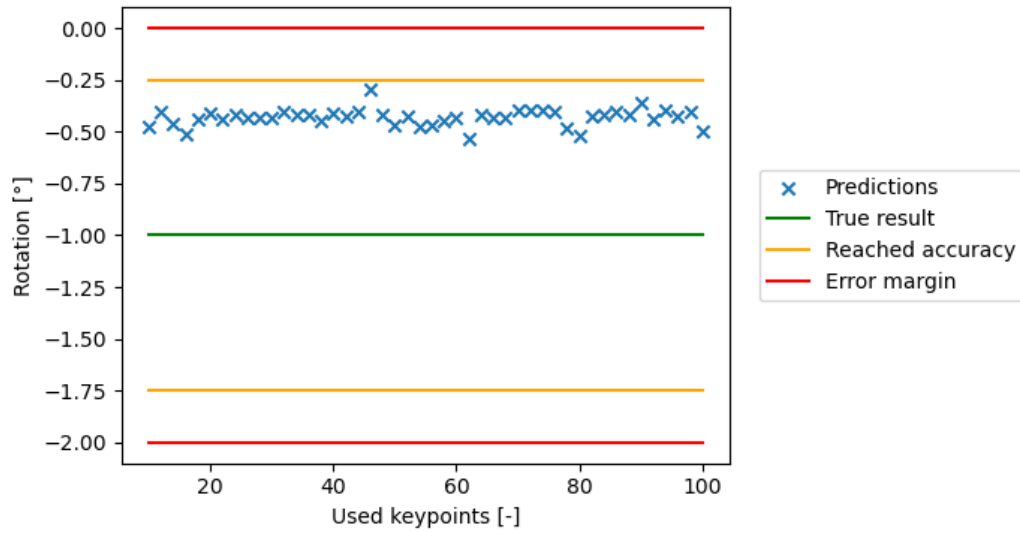


Figure 35: Rotation calculated around the Z-axis 1.

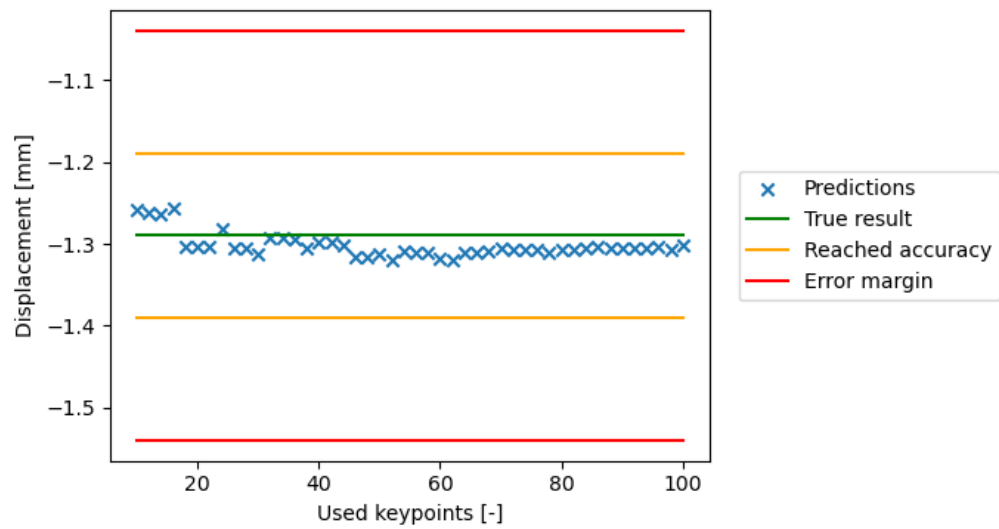


Figure 36: Displacement calculated along the X-axis 2.

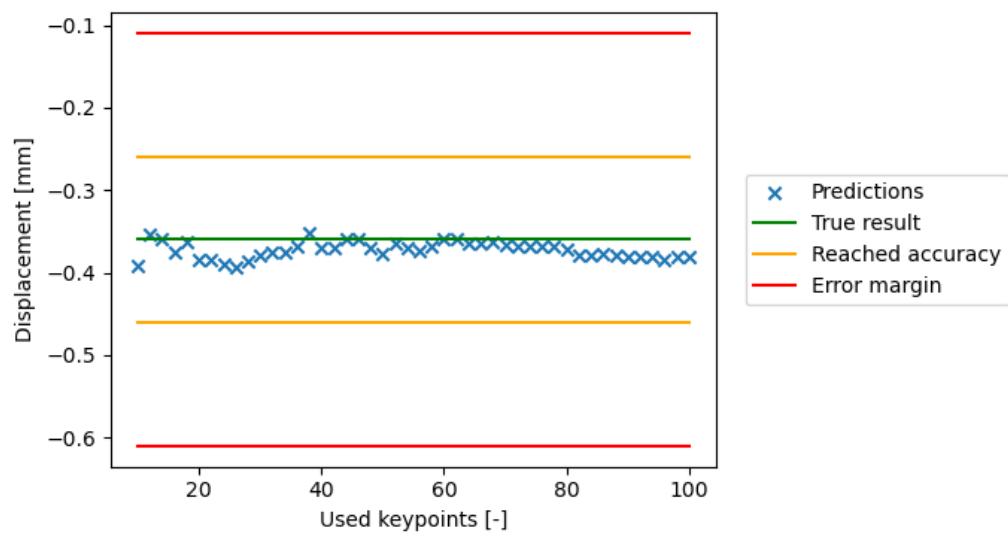


Figure 37: Displacement calculated along the Y-axis 2.

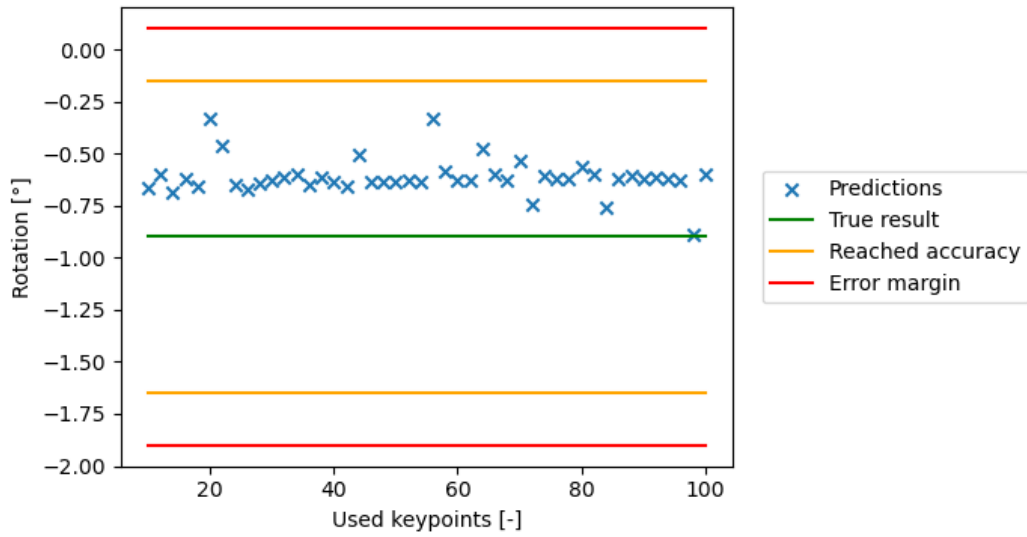


Figure 38: Rotation calculated around the Z-axis 2.

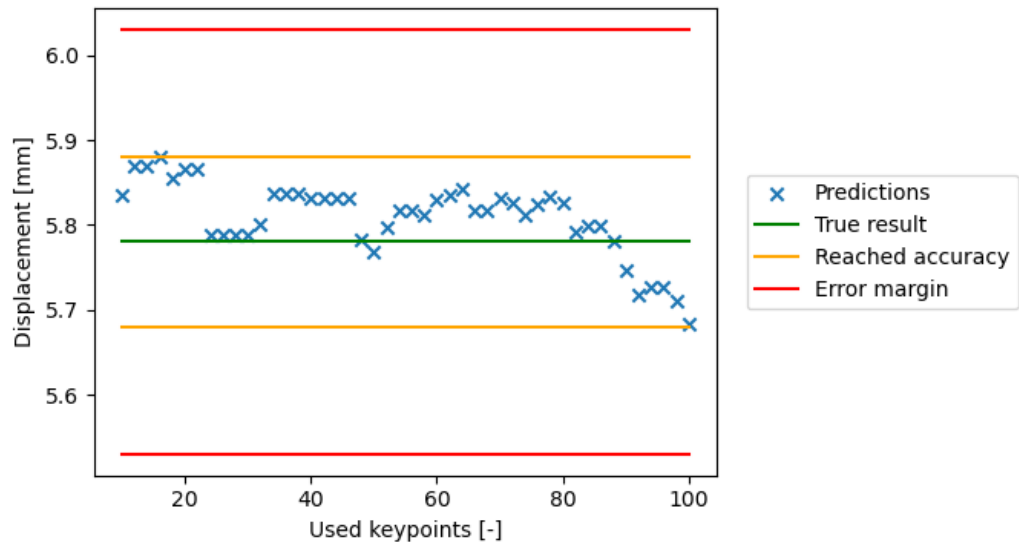


Figure 39: Displacement calculated around the X-axis 3.

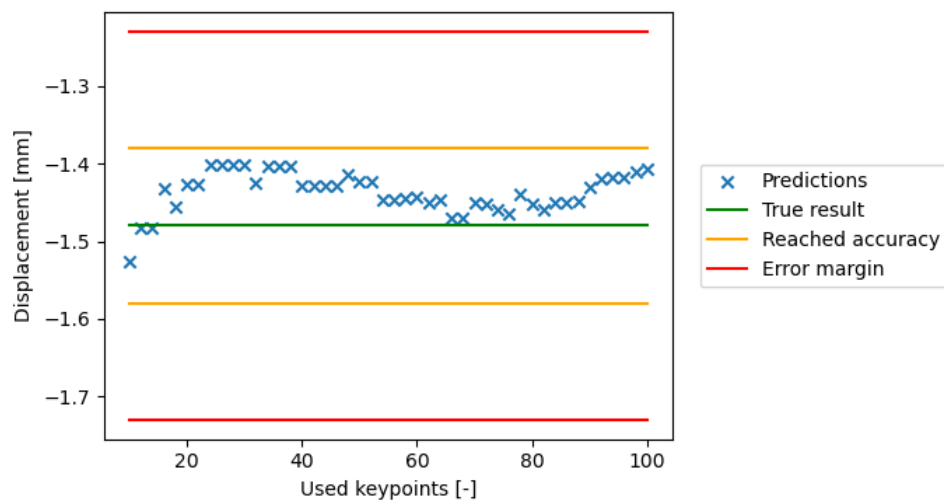


Figure 40: Displacement calculated around the Y-axis 3.

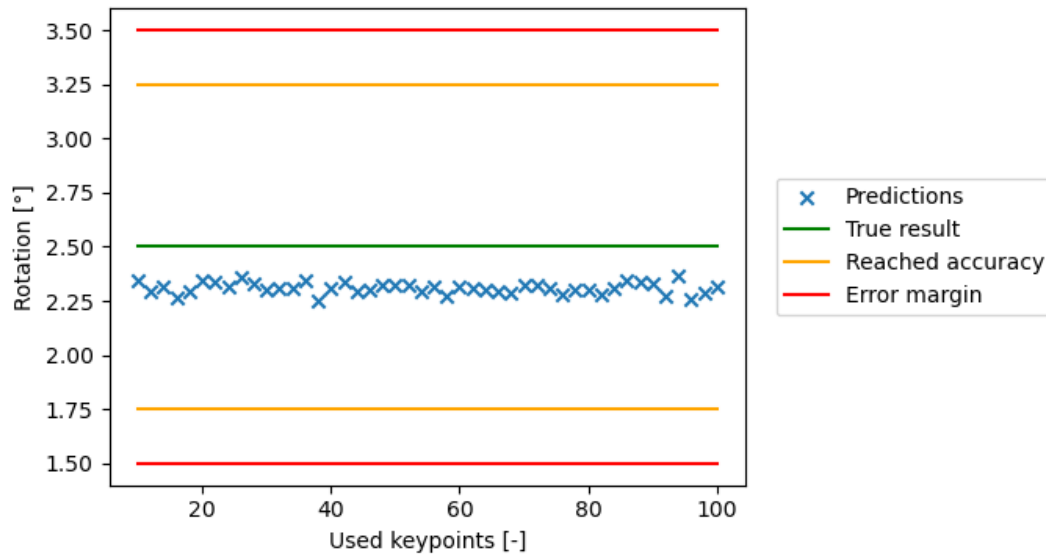


Figure 41: Rotation calculated around the Z-axis 3.

As can be seen from Figures 33 to 41, the 30 to 40 range for the number of keypoints to be used is indeed a good choice. In this range, all possibilities are exceptionally good for both rotation and displacement, but otherwise, there is no greater deviation than allowed for either displacement or rotation for any of the objects. In some places, it can be observed that the estimated values change stepwise depending on the number of keypoints used. The reason for this is that as I increase the number of keypoints used, it may not significantly change the dispersion for a long time, so no really poor estimation points pass through the second quality control filter. However, once a really bad point is included with this increase, the dispersion suddenly jumps, allowing more points to pass through the second quality control, resulting in a significant change in estimation. Nonetheless, these values remain within the error margin.

In addition, the estimation typically moves well within the allowed range with a small offset error. I was able to achieve not only the targeted 0.25 mm displacement estimation accuracy but also the outstanding 0.1 mm accuracy, and in terms of rotation estimation, I managed to implement 0.75° accuracy instead of 1°. In the development of these values, in most cases, a clear noiselessness and continuity can be observed, which confirms that image noise does not cause significant problems in determining the correction. This, incidentally, not only validates the correct operation of the software, distortion corrections, the adequacy of segmentation, and corrections by keypoints, but also the appropriate hardware selection in the case of the camera and optics, as well as the appropriate hardware design in the case of the camera holder. The system can be considered robust, as it always finds at least twice as many keypoints as necessary, and in every case, it provides an estimation within the error margin for all tests performed. It is worth noting that these tests were conducted under conditions identical to those of real measurements.

6 Summary and opportunities for further development

In the framework of my thesis, I have presented the problem of automating T3Ster measurements and identified methods that increase the efficiency of addressing challenges in this area. I designed, developed, implemented, verified, and analyzed the necessary processes analytically. I created a complex electronic device and vision system that contributes to enabling the automated operation of the internationally renowned DBC thermal characterization measurements commonly used in the automotive industry. I have been performing this measurement using traditional methods for several months, so I have gained a good understanding of it during that time. A prerequisite for robotic implementation is the availability of adequate information about the whereabouts of the objects to be measured.

To solve this problem, I designed an algorithm, performed analytical calculations, developed several hardware and software tools, selected the most suitable ones from existing tools, and worked with them. In this way, I chose a sensor, camera, and optics by performing appropriate optical calculations, selecting the economically optimal solution from several acceptable technical solutions. Then, I designed a suitable mechanical construction for mounting the camera, and for transmitting its image, I chose a suitable embedded platform and implemented a server on it. I performed two corrections on the camera image using my algorithm and the functions of the OpenCV library. The first was due to distortion, and the second was due to perspective projection. Next, I performed semantic segmentation using fully convolutional neural networks, for which I conducted thorough literature research. In this regard, I would highlight Chapters 5.4.1, 5.4.2.9, and 5.4.2.10. I prepared and manually annotated the images required for training in accordance with real measurement conditions. Precise semantic segmentation is expected to solve such a specific problem. Accordingly, my software can be considered internationally good, producing an IoU value of 98.37% on the test dataset. Finally, I performed keypoint search on the area selected with semantic segmentation, which I did using the SIFT algorithm. For filtering these keypoints by quality, I implemented a two-stage algorithm to achieve the most accurate estimation and robust operation. My developed system also shows synergy with the process of evaluating measurements, as its images can be used to view the environmental parameters of individual DBCs after thermal analysis. This provides significant assistance to my team's simulation engineers in their work.

My work is scalable and can be used for further scientific and industrial innovations and projects. In the future, I would like to measure other types of power electronics modules built on different carriers using similar techniques with minimal modifications to the existing system. In the future, I would like to try and investigate a few more things and their impact on my system. On the hardware side, this includes building a lighting system and using a polarizing filter, which could potentially improve the reliability of my developed device. On the software side, I would investigate the effects of changing further parameters, functions, and settings of my neural networks. These may include, without limitation, individual stride values, kernel size, activation function, or data augmentation. With further investigation of these, it may be possible to achieve even greater accuracy than the successfully achieved 100 μm and 0.75°, allowing for the measurement of even smaller devices than the DBCs discussed in this work.

7 References

- [1] A. Yahyaee, A. S. Bahman, and F. Blaabjerg, "A Modification of Offset Strip Fin Heatsink with High-Performance Cooling for IGBT Modules," *Applied Sciences*, vol. 10, no. 3, 2020.
- [2] Infinity Business Insights: Global Alumina DBC (Direct Bond Copper Substrate) Market Research Report 2022 (Status and Outlook), 2022.
- [3] Bosson Research: Global Market Research Reports Expert, <https://bossonresearch.com/> [Last access: 2022-10-15]
- [4] Industry Research: Global Alumina DBC (Direct Bond Copper Substrate) Market Insights an Forecast to 2028, 2022.
- [5] Market.Biz: Global Alumina DBC(Direct Bond Copper Substrate) Market By Type(Alumina Content: < 90%, Alumina Content: 90%, Alumina Content: 96%, Alumina Content: 99%, and Others), By Application, By Country, and Manufacture - Industry Segment, Competition Scenario and Forecast by 2030, 2020.
- [6] Report Mines: Global Alumina DBC (Direct Bond Copper Substrate) Market Growth 2022-2028, 2022.
- [7] MarketResearch.com: Global Alumina DBC (Direct Bond Copper Substrate) Market Research Report 2020, 2020.
- [8] Herald Keeper: Alumina DBC Direct Bond Copper Substrate Market Size, Scope and Forecast, 2022.
- [9] Xcellent Insights: Global Alumina DBC (Direct Bond Copper Substrate) Market Research Report 2022 Professional Edition, 2022.
- [10]Marketsandresearch.biz: Global Alumina DBC (Direct Bond Copper Substrate) Market 2022 by Manufacturers, Regions, Type and Application, Forecast to 2028, 2022.
- [11]VicMarketResearch: Global Alumina DBC (Direct Bond Copper Substrate) Market Outlook 2022, 2022.
- [12]H. Wang, K. Ma, and F. Blaabjerg, "Design for reliability of power electronic systems," in *Proc. 38th Annu. Conf. IEEE Ind. Electron. Soc.*, 2012, pp. 33–44.
- [13]J. Lemmens, P. Vanassche and J. Driesen, "Optimal Control of Traction Motor Drives Under Electrothermal Constraints," in *IEEE Journal of Emerging and Selected Topics in Power Electronics*, vol. 2, no. 2, pp. 249-263, June 2014.
- [14]A. Wintrich, U. Nicolai, W. Tursky, and T. Reimann, "Application manual power semiconductors," SEMIKRON International, pp. 115–129, 2011.
- [15]P. I. Prodanov, "Research of Reliability of Power Semiconductors in Dependence of Thermal Modes", *Proc. of 21th International Symposium on Electrical Apparatus and Technologies SIELA*, 2020.
- [16]C. J. M. Lasance, A. Poppe: *Thermal Management for LED Applications*, New York, USA: Springer Publishing, 2014
- [17]"Simcenter T3STER: Characterize the thermal properties of components", link: <https://www.plm.automation.siemens.com/global/en/products/simcenter/t3ster.html> , [Last access: 2022-09-20]
- [18]"Mentor Graphics - MicReD Products: T3Ster Master", link: <https://t3ster-master.software.informer.com/> , [Last access: 2022-09-20]
- [19]F. N. Masana, "A new approach to the dynamic thermal modelling of semiconductor packages", *Microelectronics Reliability*, vol. 41, pp. 901-912., 2001.
- [20]G. Fodor: *Elektromágneses terek*, pp. 163-166., Budapest, Hungary, Műegyetemi Kiadó, 2005.
- [21]G. Gróf: *Hőközlés*, pp. 10-18., 54-59., Budapest, Hungary, 1999.

- [22]A. Poppe, “The transient and multichip issues”, 11th THERMINIC Workshop, Thermal measurements and modelling, Belgirate, Italy, 2005.
- [23]V. Székely, “Identification of Networks by Deconvolution: Chances and Limits”, IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications, vol. 45., 1998.
- [24]Gy. Ábrahám, K. W. Gerőfy, A. Antal, G. Kovács: Műszaki Optika, ch. 1., 7., Budapest, Hungary, BME MOGI, 2015.
- [25]N. Sischka, “Pixel Sizes and Optics”, Automate 2017 Conference, Detroit, Michigan, USA, 2017.
- [26]Gy. Ábrahám, G. Kovács, A. Antal, Z. Németh, Á. L. Veres: Jármű Optika, ch. 1., Budapest, Hungary, BME MOGI, 2014.
- [27]“Waveshare Electronics: 8-50mm Zoom Lens”, link: https://www.waveshare.com/wiki/8-50mm_Zoom_Lens_for_Pi , [Last access: 2022-09-20]
- [28]“SONY: IMX477”, link: https://www.sony-semicon.co.jp/products/common/pdf/IMX477-AACK_Flyer.pdf , [Last access: 2022-09-20]
- [29]“Edmund Optics: (8-48mm FL), 6X Manual Zoom Video Lens”, link: <https://www.edmundoptics.eu/p/8-48mm-fl-6x-manual-zoom-video-lens/10695/> , [Last access: 2022-09-20]
- [30]“Edmund Optics: EO-10012C ½" CMOS Color USB Camera”, link: <https://www.edmundoptics.eu/p/eo-10012ccmos-color-usb-camera/22777/> , [Last access: 2022-09-20]
- [31]“Hangzhou Ai Ke Electronics: ACM117VP104013CR1”, link: <https://aico-lens.com/product/10-40mm-f1-3-12mp-p-iris-motorized-autofocus-c-mount-4x-zoom-cctv-lens-acm117vp104013cr1/> , [Last access: 2022-09-20]
- [32]“The Imaging Source: DFK 33UX226”, link: <https://www.theimagingsource.com/products/industrial-cameras/usb-3.0-color/dfk33ux226/> , [Last access: 2022-09-20]
- [33]“Raspberry Pi: HQ Camera”, link: <https://www.raspberrypi.org/products/raspberry-pi-high-quality-camera/> , [Last access: 2022-09-20]
- [34]“Raspberry Pi: 4 Model B”, link: <https://www.raspberrypi.org/products/raspberry-pi-4-model-b/> , [Last access: 2022-09-20]
- [35]P. Drap, J. Lefèvre, “An Exact Formula for Calculating Inverse Radial Lens Distortions”, Sensors, MDPI, 2016.
- [36]“OpenCV: Camera Calibration, Python Documentation”, link: https://docs.opencv.org/3.4.15/dc/dbb/tutorial_py_calibration.html , [Last access: 2022-10-04]
- [37]“OpenCV: Camera Calibration, Github”
<https://github.com/opencv/opencv/blob/master/modules/calib3d/src/calibration.cpp> , [Last access: 2022-10-04]
- [38]“Georgia Tech Computing: Computer Vision Lecture 3 Camera Calibration, 2020.”
https://www.cc.gatech.edu/classes/AY2016/cs4476_fall/results/proj3/html/kshu6/index.html , [Last access: 2022-10-04]
- [39]“OpenCV: findChessboardCorners, Documentation”, link: https://docs.opencv.org/3.4.15/d9/d0c/group_calib3d.html#ga93efa9b0aa890de240ca32b11253dd4a [Last access: 2022-10-04]
- [40]“OpenCV: cornerSubPix, Documentation”, link: https://docs.opencv.org/3.4.15/dd/d1a/group_imgproc_feature.html#ga354e0d7c86d0d9da75de9b9701a9a87e [Last access: 2022-10-04]
- [41]R. Cipolla: Computer Vision, Handout 3: Projection, University of Cambridge Engineering Part IIB, 2020

- [42]A. Likas, N. Vlassis, J. J. Verbeek, "The global k-means clustering algorithm", Pattern recognition, Elsevier, vol. 36, pp. 451–461., 2003.
- [43]X. Zheng, Q. Lei, R. Yao, Y. Gong, Q. Yin, "Image segmentation based on adaptive K-means algorithm", EURASIP Journal on Image and Video Processing, 2018.
- [44]M. Barthakur, K. K. Sarma, "Semantic Segmentation using K-means Clustering and Deep Learning in Satellite Image", 2nd International Conference on Innovations in Electronics, Signal Processing and Communication, 2019.
- [45]"OpenCV: K-means, Github"
<https://github.com/opencv/opencv/blob/master/modules/core/src/kmeans.cpp> , [Last access: 2022-10-06]
- [46]F. Pierre, J. F. Aujol, A. Bugeau, V. T. Ta, "Luminance-Hue Specification in the RGB Space", International Conference on Scale Space and Variational Methods in Computer Vision, pp. 413-424., 2015.
- [47]X. Li, Y. Wu, "Image object detection algorithm based on improved Gaussian mixture model", Journal of Multimedia, pp. 152-158., 2014.
- [48]C. Clausen, H. Wechsler, "Color image compression using PCA and backpropagation learning", Pattern Recognition, vol. 33., pp. 1555-1560., 2000.
- [49]"OpenCV: floodFill, Github"
<https://github.com/opencv/opencv/blob/master/modules/imgproc/src/floodfill.cpp> [Last access: 2022-10-07]
- [50]"OpenCV: Morphological Operations, Github"
<https://github.com/egonSchiele/OpenCV/blob/master/modules/imgproc/src/morph.cpp> [Last access: 2022-10-07]
- [51]S. A. Taghanaki, K. Abhishek, J. P. Cohen, J. Cohen-Adad, G. Hamarneh, "Deep semantic segmentation of natural and medical images: a review", Springer, 2020.
- [52]F. Jiang, A. Grigorev, S. Rho, Z. Tian, Y. Fu, W. Jifara, K. Adil, S. Liu, "Medical image semantic segmentation based on deep learning", Springer, 2017.
- [53]B. Kayalibay, G. Jensen, P. van der Smagt, "CNN-based segmentation of medical imaging data", Germany, 2017.
- [54]W. T. Xiao, L. J. Chang, W. M. Liu, "Semantic segmentation of colorectal polyps with DeepLab and LSTM networks", International Conference on Consumer Electronics-Taiwan, 2018.
- [55]Y. Wu, L. Lin, J. Wang, S. Wu, "Application of semantic segmentation based on convolutional neural network in medical images", Journal of Biomedical Engineering, 2020.
- [56]R. Fan, H. Wang, P. Cai, M. Liu, "Sne-roadseg: Incorporating surface normal information into semantic segmentation for accurate freespace detection", European Conference on Computer Vision, 2020.
- [57]Z. Pan, T. Emaru, A. Ravankar, Y. Kobayashi, "Applying semantic segmentation to autonomous cars in the snowy environment", 36th Annual Conference of the Robot Society of Japan, Nagoya, 2018.
- [58]L. Sun, K. Yang, X. Hu, W. Hu, K. Wang, "Real-Time Fusion Network for RGB-D Semantic Segmentation Incorporating Unexpected Obstacle Detection for Road-Driving Images", IEEE Robotics and Automation Letters, vol. 5., 2020.
- [59] B. Li, S. Liu, W. Xu, W. Qiu, "Real-time object detection and semantic segmentation for autonomous driving", Proceedings of the Multispectral Image Processing and Pattern Recognition: Automatic Target Recognition and Navigation, 10608, International Society for Optics and Photonics, 2018.

- [60]S. Liu, M. Li, M. Li, Q. Xu, "Research of animals image semantic segmentation based on deep learning", *Concurrency and Computation: Practice and Experience*, vol. 32, 2018.
- [61]R. Sahu, "Semantic body parts segmentation for quadrupedal animals ", *IEEE International Conference on Systems, Man and Cybernetics*, 2016.
- [62]I. Alonso, A. Cambra, A. Munoz, T. Treibitz, A. C. Murillo, "Coral-Segmentation: Training Dense Labeling Models With Sparse Ground Truth", *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 2874-2882, 2017.
- [63]G. Brazil, X. Yin, X. Liu, "Illuminating pedestrians via simultaneous detection & segmentation", *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 4950-4959, 2017.
- [64]G. Cheng, J. Y. Zheng, "Semantic Segmentation for Pedestrian Detection from Motion in Temporal Domain", *IEEE 25th International Conference on Pattern Recognition*, 2020.
- [65]X. Tao, D. Zhang, W. Ma, X. Liu, D. Xu, "Automatic Metallic Surface Defect Detection and Recognition with Convolutional Neural Networks", *Applied Sciences*, vol. 8, 2018.
- [66]V. V. Kniaz, "Conditional GANs for semantic segmentation of multispectral satellite images", *Proceedings of Image and Signal Processing for Remote Sensing XXIV*, Berlin, Germany, 2018.
- [67]M. Wurm, T. Stark, X. X. Zhu, M. Weigand, H. Taubenböck, "Semantic segmentation of slums in satellite images using transfer learning on fully convolutional neural networks", *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 150, pp. 59-69., 2019.
- [68]Y. Guo, Y. Liu, T. Georgiou, M. S. Lew, "A review of semantic segmentation using deep neural networks", *International Journal of Multimedia Information Retrieval*, vol. 7, pp. 87–93., 2018.
- [69]F. Lateef, Y. Ruichek, "Survey on semantic segmentation using deep learning techniques", *Neurocomputing*, vol. 338, pp. 321-348, 2019.
- [70]B. Li, Y. Shi, Z. Qi, Z. Chen, "A Survey on Semantic Segmentation", *IEEE International Conference on Data Mining Workshops (ICDMW)*, 2018.
- [71]H. Yu, Z. Yang, L. Tan, Y. Wang, W. Sun, M. Sun, Y. Tang, "Methods and datasets on semantic segmentation: A review ", *Neurocomputing*, vol. 304, pp. 82-103, 2018.
- [72]S. Hao, Y. Zhou, Y. Guo, "A Brief Survey on Semantic Segmentation with Deep Learning", *Neurocomputing*, vol. 406, pp. 302-321, 2020.
- [73]A. G. Garcia, S. O. Escolano, S.O. Oprea, V. V. Martinez, and J. G. Rodriguez, "A Review on Deep Learning Techniques Applied to Semantic Segmentation", 2017., arXiv:1704.06857. [Online] link: <https://arxiv.org/abs/1704.06857> [Last access: 2022-09-24]
- [74]R. Kestur, S. Farooq, R. Abdal, E. Mehraj, O. S. Narasipura, M. Mudigere, "UFCN: a fully convolutional neural network for road extraction in RGB imagery acquired by remote sensing from an unmanned aerial vehicle", *Journal of Applied Remote Sensing*, 2018.
- [75]Z. Wu, Y. Gao, L. Li, J. Xue, Y. Li, "Semantic segmentation of high-resolution remote sensing images using fully convolutional network with adaptive threshold", *Connection Science*, vol. 31, 2019.
- [76]D. L. Torres, R. Q. Feitosa, P. N. Happ, L. E. Cué, L. Rosa, J. M. Junior, J. Martins, P. O. Bressan, W. N. Gonçalves, V. Liesenberg, "Applying Fully Convolutional Architectures for Semantic Segmentation of a Single Tree Species in Urban Environment on High Resolution UAV Optical Imagery", *Sensors*, vol. 20, 2020.
- [77]W. Bai, O. Oktay, M. Sinclair, H. Suzuki, M. Rajchl, G. Tarroni, B. Glocker, A. King, P. M. Matthews, D. Rueckert, "Semi-supervised Learning for Network-Based Cardiac MR Image Segmentation", pp. 253-260., *20th International Conference Quebec City, QC, Canada, Proceedings, Part II*, 2017.

- [78]C. V. Dung, L. D. Anh, "Autonomous concrete crack detection using deep fully convolutional neural network", *Automation in Construction*, vol. 99., pp. 52-58.
- [79]X. Ma, X. Deng, L. Qi, Y. Jiang, H. Li, Y. Wang, X. Xing, "Fully convolutional network for rice seedling and weed image segmentation at the seedling stage in paddy fields", *PloS one*, 2019.
- [80]L. Tai, H. Ye, Q. Ye, M. Liu, "PCA-aided fully convolutional networks for semantic segmentation of multi-channel fMRI", *International Conference on Advanced Robotics (ICAR)*, 2017.
- [81]G. Chen, X. Zhang, Q. Wang, F. Dai, Y. Gong, K. Zhu, "Symmetrical Dense-Shortcut Deep Fully Convolutional Networks for Semantic Segmentation of Very-High-Resolution Remote Sensing Images", *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 2018.
- [82]J. Fu, Y. Wang, H. Lu, "Stacked deconvolutional network for semantic segmentation", *IEEE Transactions on Image Processing*, 2017.
- [83]Y. Wang, J. Liu, Y. Li, J. Yan, H. Lu, "Objectness-aware semantic segmentation", *Proceedings of the ACM on Multimedia Conference*, ACM, pp. 307–311., 2016.
- [84]H. Noh, S. Hong, B. Han, "Learning deconvolution network for semantic segmentation", *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1520–1528., 2015.
- [85]S. Jégou, M. Drozdal, D. Vazquez, A. Romero, Y. Bengio, "The one hundred layers tiramisu: fully convolutional densenets for semantic segmentation", *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, IEEE, pp. 1175–1183., 2017.
- [86]V. Badrinarayanan, A. Kendall, R. Cipolla, "Segnet: a deep convolutional encoder-decoder architecture for image segmentation", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39., no. 12., 2017.
- [87] J. Long, E. Shelhamer, T. Darrell, Fully convolutional networks for semantic segmentation, "Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition", pp. 3431–3440., 2015.
- [88]"Papers with Code: Semantic Segmentation on PASCAL VOC 2012 test", link: <https://paperswithcode.com/sota/semantic-segmentation-on-pascal-voc-2012> , [Last access: 2022-09-25]
- [89]"Papers with Code: CamVid (Cambridge-driving Labeled Video Database)", link: <https://paperswithcode.com/sota/semantic-segmentation-on-pascal-voc-2012https://paperswithcode.com/dataset/camvid> , [Last access: 2022-09-25]
- [90]"Papers with Code: Semantic Segmentation on PASCAL VOC 2011", link: <https://paperswithcode.com/sota/semantic-segmentation-on-pascal-voc-2011> , [Last access: 2022-09-25]
- [91]Y. Li, S. Liu, C. Li, Y. Zheng, C. Wei, B. Liu, Y. Yang, "Automated defect detection of insulated gate bipolar transistor based on computed laminography imaging", *Microelectronics Reliability*, vol. 115., 2020.
- [92]W. Shi, Z. Lu, W. Wu, H. Liu, "Single-shot detector with enriched semantics for PCB tiny defect detection", *The Journal of Engineering*, 2020.
- [93]Z. Yang, R. Dong, H. Xu, J. Gu, "Instance Segmentation Method Based on Improved Mask R-CNN for the Stacked Electronic Components", *Electronics*, 2020.
- [94]J. Lian, L. Wang, T. Liu, X. Ding, Z. Yu, "Automatic visual inspection for printed circuit board via novel Mask R-CNN in smart city applications", *Sustainable Energy Technologies and Assessments*, vol 44., 2021.
- [95]D. Li, C. Li, C. Chen, Z. Zhao, "Semantic Segmentation of a Printed Circuit Board for Component Recognition Based on Depth Images", *Sensors*, vol. 20., 2020.
- [96]"Pytorch: Documentation", link: <https://pytorch.org/docs/stable/index.html> [Last access: 2022-10-08]

- [97] “Torchvision: Documentation”, link: <https://pytorch.org/vision/stable/index.html> [Last access: 2022-10-08]
- [98] “Pillow: Documentation”, link: <https://pillow.readthedocs.io/en/stable/> [Last access: 2022-10-08]
- [99] “Numpy: Documentation”, link: <https://numpy.org/doc/> [Last access: 2022-10-08]
- [100] “Matplotlib: Documentation”, link: <https://matplotlib.org/> [Last access: 2022-10-08]
- [101] “OS: Documentation”, link: <https://docs.python.org/3/library/os.html> [Last access: 2022-10-08]
- [102] “Glob: Documentation”, link: <https://docs.python.org/3/library/glob.html> [Last access: 2022-10-08]
- [103] X. Sun, P. Zhang, D. Wang, Y. Cao and B. Liu, "Colorectal Polyp Segmentation by U-Net with Dilation Convolution," 2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA), pp. 851-858, 2019.
- [104] J. Thanaki: Python natural language processing, pp. 397-398., 2017.
- [105] M. Szemenyei: Számítógépes Látórendszerek, pp. 110-111. 2021.
- [106] M. Szemenyei: Számítógépes Látórendszerek, pp. 129-130. 2021.
- [107] H. Yang, J. Liu, H. Sun, H. Zhang: “PACL: piecewise arc cotangent decay learning rate for deep neural network training”, IEEE Access, vol. 8., 2020.
- [108] “NVIDIA: CUDA Toolkit Release Notes”, link: <https://docs.nvidia.com/cuda/cuda-toolkit-release-notes/index.html> [Last access: 2022-10-09]
- [109] L. Feng, S. Shu, Z. Lin, F. Lv, L. Li, B. An, “Can cross entropy loss be robust to label noise? ”, Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, 2020.
- [110] Z. Zhang, M. R. Sabuncu, “Generalized cross entropy loss for training deep neural networks with noisy labels”, 32nd Conference on Neural Information Processing Systems, Montréal, Canada, 2018.
- [111] D. P. Kingma, J. Ba, “Adam: A method for stochastic optimization”, ICLR, 2015.
- [112] I. K. M. Jais, A. R. Ismail, S. Q. Nisa, “Adam optimization algorithm for wide and deep neural network”, Knowledge Engineering and Data Science, 2019.
- [113] L. Bottou, “Stochastic Gradient Descent Tricks,” in Neural Networks: Tricks of the Trade: Second Edition, G. Montavon, G. B. Orr, and K.-R. Müller, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 421–436, 2012.
- [114] “Alpha: Make Sense”, link: <https://www.makesense.ai/> , [Last access: 2022-10-09]
- [115] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kegl, “Algorithms for Hyper-Parameter Optimization,” Proc. Neural Information and Processing Systems, 2011.
- [116] M. Feurer and F. Hutter, “Hyperparameter optimization,” in Automated machine learning, Springer, Cham, pp. 3–33, 2019.
- [117] P. I. Frazier, “A tutorial on Bayesian optimization,” arXiv preprint arXiv:1807. 02811, 2018.
- [118] Y. Bengio, “Gradient-based optimization of hyperparameters,” Neural computation, vol. 12, no. 8, pp. 1889–1900, 2000.
- [119] D. Simon, Evolutionary optimization algorithms. John Wiley & Sons, 2013.
- [120] M. Massaoudi, H. Abu-Rub, S. S. Refaat, M. Trabelsi, I. Chihi and F. S. Oueslati, “Enhanced Deep Belief Network Based on Ensemble Learning and Tree-Structured of Parzen Estimators: An Optimal Photovoltaic Power Forecasting Method, ” in IEEE Access, vol. 9, pp. 150330-150344, 2021.
- [121] J. Liang et al., “Intelligent fault diagnosis of rotating machinery using lightweight network with modified tree-structured parzen estimators,” IET Collaborative Intelligent Manufacturing, 2022.
- [122] G. Rong et al., “Comparison of Tree-Structured Parzen Estimator Optimization in Three Typical Neural Network Models for Landslide Susceptibility Assessment,” Remote Sensing, vol. 13, no. 22, 2021.

- [123] H.-P. Nguyen, J. Liu, and E. Zio, "A long-term prediction approach based on long short-term memory neural networks with automatic parameter optimization by Tree-structured Parzen Estimator and applied to time-series data of NPP steam generators," *Applied Soft Computing*, vol. 89, p. 106116, 2020.
- [124] S. M. Hernandez and E. Bulut, "WiFi Sensing on the Edge: Signal Processing Techniques and Challenges for Real-World Systems," *IEEE Communications Surveys & Tutorials*, pp. 1–1, 2022.
- [125] M. Zhao and J. Li, "Tuning the hyper-parameters of CMA-ES with tree-structured Parzen estimators," *2018 Tenth International Conference on Advanced Computational Intelligence (ICACI)*, pp. 613-618, 2018.
- [126] S. Yoo, M. A. Haider, and F. Khalvati, "Estimating optimal depth of vgg net with tree-structured parzen estimators," *Journal of Computational Vision and Imaging Systems*, vol. 3, no. 1, 2017.
- [127] Z. Wang, L. Wang, Q. Liang, W. Luo, Q. Gong and S. Li, "Research on Automated Reinforcement Learning: based on Tree-structured Parzen Estimators and Median Pruning," *2021 33rd Chinese Control and Decision Conference (CCDC)*, pp. 5461-5465, 2021.
- [128] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A Next-generation Hyperparameter Optimization Framework," in *Proceedings of the 25rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.
- [129] B. Resch, "Mixtures of gaussians," *A Tutorial for the Course Computational Intelligence. Signal Processing and Speech Communication Laboratory*, vol. 24, 2010.
- [130] X. Ying, "An overview of overfitting and its solutions," in *Journal of physics: Conference series*, vol. 1168, p. 022022, 2019.
- [131] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints", *International Journal of Computer Vision*, 2004.
- [132] "OpenCV: SIFT, Documentation", https://docs.opencv.org/3.4.15/d7/d60/classcv_1_1SIFT.html , [Last access: 2022-10-09]
- [133] "OpenCV: BFMatcher, Documentation", https://docs.opencv.org/4.5.3/d3/da1/classcv_1_1BFMatcher.html , [Last access: 2022-1-09]