

Copyright

by

Varun Soni

2020

**The Thesis Committee for Varun Soni
Certifies that this is the approved version of the following Thesis:**

A machine learning optical system to ensure that human assembly technicians use the specified bolt tightening sequence in assembly line manufacturing.

**APPROVED BY
SUPERVISING COMMITTEE:**

Preston S. Wilson, Supervisor

Richard H. Crawford

A machine learning optical system to ensure that human assembly technicians use the specified bolt tightening sequence in assembly line manufacturing.

by

Varun Soni

Thesis

Presented to the Faculty of the Graduate School of
The University of Texas at Austin
in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science in Engineering

The University of Texas at Austin

May 2020

Acknowledgements

I wish to express my sincere appreciation to my supervisor, Dr. Preston Wilson, for taking me on as his student and providing me the valuable guidance to complete this project. He encouraged me to be professional and do the right thing even when the road got tough. Without his persistent help, the goal of this project would not have been realized. I would like to express my gratitude to Dr. Richard Crawford for his steadfast support and mentorship throughout the past two years. Finally, I wish to acknowledge the support and great love of my grandfather, K.K. Soni; my mother, Vaneeta; and my sisters, Priyanka and Hansu.

Abstract

A machine learning optical system to ensure that human assembly technicians use the specified bolt tightening sequence in assembly line manufacturing.

Varun Soni, MSE

The University of Texas at Austin, 2020

Supervisor: Preston S. Wilson

In large number of applications, the mechanical fasteners that are used to assemble the parts of a system must be tightened in a specific sequence to achieve the desired distribution of the load across the population of bolts. Failure to follow the sequence results in an undesired load distribution; this phenomenon is known as bolt crosstalk. Assembly personnel often fail to follow this sequence for a variety of reasons, resulting in over- or under-torqueing of bolts in the final assembly, which can lead to undesired system performance. There is currently no system or device that can ensure that a human operator follows a specified bolt tightening sequence while using a hand-held tool and thereby avoid bolt crosstalk. In this research, a system that constrains the operator to follow the specified tightening sequence was developed and tested. It utilizes a small tool-mounted camera to generate images of the bolt pattern and the relative location of the tool, and a machine

learning algorithm to alert the operator if the tool is being brought to the wrong position. The developed software can detect all the bolt positions accurately by using a unique feature associated with them. The average of probabilities of detecting a position in different lightning conditions is more than 85%.

Table of Contents

List of Tables	ix
List of Figures	x
Chapter 1: Introduction	1
1.1 Bolt crosstalk	2
1.2 Project domain	4
1.3 Problem statement.....	4
1.4 Industry analysis	5
1.5 Thesis organization	7
Chapter 2: Literature review	8
2.1 Radio frequency identification (RFID) system.....	8
2.2 Infrared system.....	9
2.3 Vision system.....	11
Chapter 3: Deep learning and software development	13
3.1 Convolutional neural network (CNN)	13
3.2 Advantages of CNN over other methods	16
3.3 Advantages of CNN over other methods.....	17
3.3.1 Convolution layers	17
3.3.2 Pooling/subsampling layers	19
3.3.3 Advantages of CNN over other methods	20
3.3.4 Advantages of CNN over other methods	21
3.4 Supervised machine learning and software development.....	23
3.5 Software requirements	24

3.6 Software details.....	26
3.6.1 Camera display.....	27
3.6.2 Snapshot/push button	27
3.6.3 Train model	33
Chapter 4: Testing and results.....	45
4.1 Detailed description of software testing	45
4.2 Software testing under different lightening conditions.....	50
4.3 Summary of test results.....	55
Chapter 5: Conclusion and recommendation	57
5.1 Conclusion	57
5.2 Improvements and recommendation	58
Appendices.....	59
Appendix A PC Specifications	59
Appendix B Arduino UNO Specifications	60
Appendix C Recommended PC Specifications	61
References.....	62

List of Tables

Table 1:	Car assembly shop cost savings analysis.....	7
Table 2:	Summary of test results.....	56

List of Figures

Figure 1:	Bolt Crosstalk.....	3
Figure 2:	Examples of bolt tightening sequence	3
Figure 3:	Exploded view of a car engine.....	6
Figure 4:	Tracking of a bolt using an RFID system	9
Figure 5:	Tool tracking using an IR system	10
Figure 6:	Screenshots from the Opti-track Tracking Tool Software.....	10
Figure 7:	Feature pattern recognition associated with each fastener (steering rack) ...	12
Figure 8:	Neural Network.....	13
Figure 9:	Biological neuron and its mathematical model.....	14
Figure 10:	Typical Block Diagram of CNN.....	16
Figure 11:	Pictorial representation of convolution process.....	18
Figure 12:	Pictorial representation of max pooling & average pooling	19
Figure 13:	Pictorial representaion of ReLU functionality	20
Figure 14:	Pictorial representaion of ReLU functionality	21
Figure 15:	Pictorial representaion of ReLU functionality	22
Figure 16:	Main screen of GUI of software.....	26
Figure 17:	Unique characteristic related to encircled bolt.....	27
Figure 18:	Flowchart of overall system	29
Figure 19:	Flowchart of developed system.....	30
Figure 20:	Screenshot showing output and temp folders	31
Figure 21:	LEDs on NOT OK and OK outputs.....	32
Figure 22:	Result file in output folder.....	33

Figure 23:	Train Model GUI.....	33
Figure 24:	Number of unique bolt positions.....	34
Figure 25:	Folders created in dataset.....	34
Figure 26:	Data to be stored in given folder.....	35
Figure 27:	Clicked picture in given folder number.....	36
Figure 28:	Clicked picture in given folder number.....	37
Figure 29:	Before random transformations of dataset.....	38
Figure 30:	Dataset after image transformations.....	38
Figure 31:	Dataset after image transformations.....	39
Figure 32:	Dataset after image transformations.....	40
Figure 33:	Rectified Linear (ReLU) function.....	41
Figure 34:	Dataset after image transformations.....	42
Figure 35:	Dataset after image transformations.....	42
Figure 36:	Output of results after each epoch.....	43
Figure 37:	Plot showing increase in accuracy with each epoch.....	43
Figure 38:	Plot showing overfitting of data.....	44
Figure 39:	Arduino UNO with different bolt positions.....	46
Figure 40:	Examples of position and unique characteristics.....	46
Figure 41:	Arduino with push button to click picture.....	47
Figure 42:	Clicked images of all positions.....	48
Figure 43:	Output in python console.....	49
Figure 44:	Screenshot of output text file.....	49
Figure 45:	Case 1: Brightness = -100%.....	50
Figure 46:	Case 1: Brightness = -100%.....	51
Figure 47:	Case 2: Brightness = -75%.....	51

Figure 48:	Case 3: Brightness = -50%	52
Figure 49:	Case 4: Brightness = -25%	52
Figure 50:	Case 5: Brightness = 25%	53
Figure 51:	Case 6: Brightness = 50%	53
Figure 52:	Case 7: Brightness = 75%	54
Figure 53:	Case 8: Brightness = 100%.....	54
Figure 54:	Chart showing summary of test results	56

Chapter 1: Introduction

There are five major methods for assembling parts in any assembly: Mechanical Assembly, Welding, Spot Welding, Riveting and Brazing/Soldering. Number of parts, functionality of the part and other factors contribute in choosing one method over the other. A summary of all the methods is as follows:

- **Welding:** Welding is the process of joining materials by using high heat to melt the parts together and letting them cool, causing fusion of material. It is mainly suited for assemblies that are permanent and need structural strength.
- **Spot Welding:** Spot Welding welds two or more metal sheets by applying pressure and heat to the weld area. It is not as permanent as the welding process mentioned above, but is more permanent than mechanical assembly and is less expensive than regular welding.
- **Riveting:** This is a forging process that joins two metal part by using a metal fastener known as a rivet. This process is mainly used for assemblies subjected to fluctuating temperature and pressure. It is cheaper and has less strength than a weld.
- **Soldering/Brazing:** A filler metal is melted to a specific temperature to join two components together. Soldered joints are not as strong as welded joints.
- **Mechanical Assembly:** This is the process of assembling multiple parts together by using fasteners such as nuts, bolts, screws, etc. It finds applications in assemblies which are not permanent and have parts which need maintenance and replacement due to wear and tear.

Mechanical assembly is still a very common method used in manufacturing. The aim of this thesis is to study two of the major the problems associated with mechanical assembly, and to provide solutions for them:

1. Bolt crosstalk
2. Identification of specific bolt

1.1 Bolt Crosstalk

In many cases, more than one bolt is used to assemble two parts together. When one of the bolts in the group is tightened, this changes the force in the other bolts due to the elasticity of the material. This phenomenon is known as bolt crosstalk. In Fig.1(A), two outer bolts (colored green) have been tightened already. Tightening the center bolt (colored red in B) results in further compression of the material (colored yellow) under the outer bolts and thus decreases the tension in the outer bolts. This tension is known as bolt preload and is the result of tightening of bolts. This preload is needed so that the joint can sustain much higher external loads as compared to a joint with loosely tightened bolts. Further, proper spatial distribution of this preload is often required. To minimize the effects of bolt crosstalk, the bolts should be tightened in a specific sequence, as shown in Fig. 2 for two different shapes.

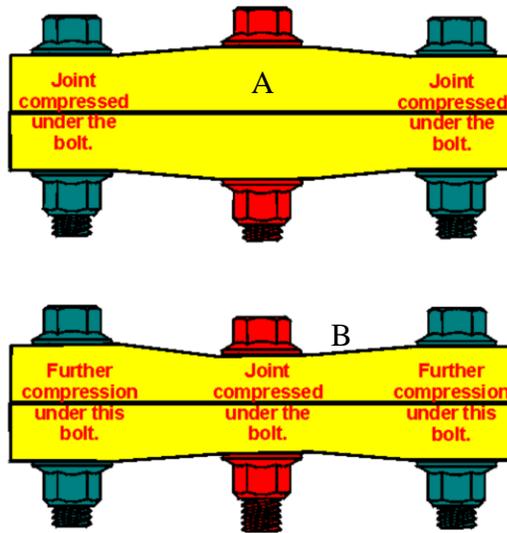


Figure 1. Bolt Crosstalk (adapted from Ref. 1)

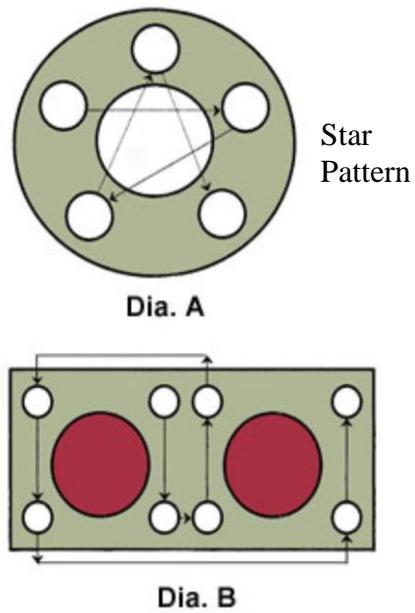


Figure 2. Examples of bolt tightening sequences (adapted from Ref. 2).

1.2 Project Domain

While working as a manufacturing engineer at India's largest car manufacturing company, the author was responsible for DC tools in the assembly shop and he observed that the operators do not follow the tightening sequence defined in company's work instruction sheets, which results in decrease in quality of the product. The purpose of this project is to develop a solution that can be used to map the torque value of each bolt and ensure the proper tightening sequence is followed to prevent bolt crosstalk. The research is focused on developing a convolutional neural network (CNN) that can identify a bolt, rather than developing the entire product, which includes tools, torque wrenches, electronics and hardware. This is because existing tightening solutions can be easily integrated with the solution developed. Upon completing development of software, testing was conducted on Arduino's base plate to see if a bolt can be identified based on its unique features.

1.3 Problem Statement

Tools that have the capability of logging torque data are used for tightening critical fasteners across all areas of manufacturing. The problem is that there can be many similar bolts to be tightened to different specified torques, by a particular human operator working at just one station. There is no device on these tools that can identify each fastener; therefore it becomes impossible to map torque values accurately to each fastener. The tightening sequence can be specified, but there is no way to guarantee that the operator will follow the sequence until after it is done.

During the author's experience as a manufacturing engineer (refer section 1.2), it has been observed that the specified tightening pattern is not followed by human operators

due to many reasons, such as loss of concentration, laziness, repetitive nature of work, etc. To overcome these issues, manufacturers often employ an additional human operator to manually check torque values on each bolt. This torque check process provides zero value added to the manufacturing process and results in increased cost and reduced productivity. Another problem of deploying a manual torque check is that torque wrenches cannot detect if a bolt has been over torqued and over torquing can result in damage to the threads on bolts, parts and nuts. Therefore, a compact system is needed which can be fitted onto any hand-operated tightening tool and can also be interfaced with the tool controller so that torque values can be mapped to individual fasteners. The tool must constrain the operator to follow the specified tightening sequence, thus resulting in elimination of a post-assembly torque check and elimination of the bolt crosstalk problem.

1.4 Industry Analysis

Although the aforementioned problems exist in many manufacturing sectors ranging from electronics to oil refineries, this study focused on the automotive industry. Fig. 3 shows the assemblies (boxed in red) in a car engine where a sequence of bolt tightening is important in order to prevent bolt crosstalk (refer section 1.2). The tightening sequence is defined in a work instruction sheet for assembly of the oil pan cover, the water pump cover, the cylinder head cover, the intake manifold and the exhaust manifold.

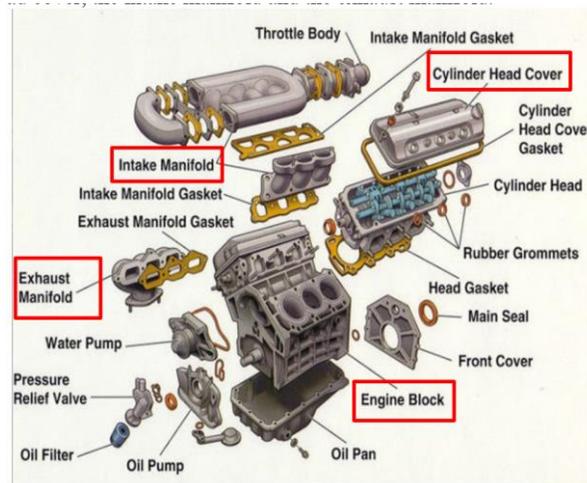


Figure 3. Exploded view of a car engine (adapted from Ref. 3).

Table 1 summarizes the number of workers responsible for manual torque checking in one of the assembly facilities at one of India's largest car-manufacturing companies (refer section 1.2). Total non-value added costs associated with bolt tightening are 6M INR (79290 USD). The aim is to develop a solution which will be less than 5K USD, so that it is cost effective and can be deployed by developing countries as well.

S. No	Area of Work	Number of workers responsible for manual torque checking
1	Trim line	3
2	Chassis line	5
3	Final Assembly line	0
	Total	8
	Number of Shifts	2
	No. of production days/year	280
	Number of hours/shift	8
	No. of operators doing manual torque check/year	35840
	Average cost/hour (INR)	170
	Total non-value cost annually (INR)	6,092,800

Table 1: Car assembly shop cost savings analysis.

1.5 Thesis Organization

This thesis is organized into three major chapters, followed by a conclusion. Chapter 2 is a review of the literature, focused on the past research that has been conducted to solve the problem of crosstalk. This background informed the development of the solution. Chapter 3 discusses convolution neural networks (CNN) and image recognition using CNN, and gives an overview of the software that has been developed. Chapter 4 discusses experiments conducted with the program, including performance of the algorithm with respect to accuracy. Chapter 5 summarizes the research and provides suggestions for improving the system.

Chapter 2: Literature Review

Significant research has been done in the past to solve the problem of workers not following the bolt tightening sequence on a station. A radio frequency identification (RFID) method has been used to match the tool position with fastener position (Vukelic et al., 2011). Research on an indoor tracking method using infrared (IR) has been done to locate bolt position and orientation in 3D space (Rusli and Luscher, 2012). Machine vision is widely used in quality inspection systems, assembly misalignment, and detection of presence of fasteners. (Killing et al., 2009).

2.1 Radio Frequency Identification (RFID)

In the method by Vukelic et al., an RFID system (Fig. 4, boxed in red) is mounted on the top of the assembly tool and RFID tags can be fixed on the part itself or on the fixture close to the desired locations. Each RFID tag has a unique identifier (UID). When the operator takes the tool to tighten the bolt, the RFID reader mounted on the tool reads the UID of the RFID tag closest to the reader and sends this data to a PC, which compares it to the UID for that bolt position. Power is supplied to the tool only when the bolt position is correct. The main disadvantage of this system is that RFID readers are bulky and cannot be mounted on the top of many tools without compromising the ergonomics of the system. In addition, the RFID system functions poorly when the bolt separation distance is too small and it is no longer possible to distinguish the different UIDs.

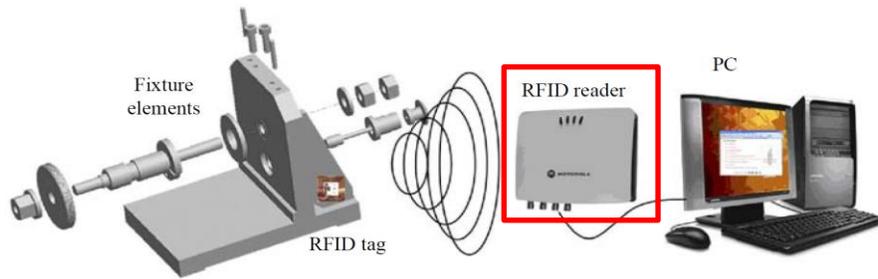


Figure 4. Tracking of a bolt using an RFID system (adapted from Ref. 4).

2.2 Infrared (IR) system

Infrared (IR) systems can track the 3D location of a bolt using optical triangulation techniques (Kraus, 2004; Luhmann et al., 2006). Reflective IR markers coated with retroreflective material are pasted on the tool. Usually, triangulation can be achieved with two IR cameras. These IR cameras illuminate the volume with IR radiation and track the locations of reflective IR markers. If the reflective markers are within the specified distance, then a signal is sent to the controller and power is applied to the tool to tighten the bolts (Fig. 5). This solution is capable of a tracking accuracy of a few millimeters. A key shortcoming of the IR tracking method is that it requires line-of-sight between the camera and the tracked object and in practice, there is often the possibility of an obstacle in the line of sight. In addition, IR tracking fails in the cases where the fastening process is totally enclosed by a shell, such as a person working inside a car body. One solution is to install more cameras but this increases the cost of overall system. Fig. 6 shows the screen of a commercially-available system, the Optitrack Tracking Software. This software is used to study the volume that will be covered by mounting cameras at different locations.

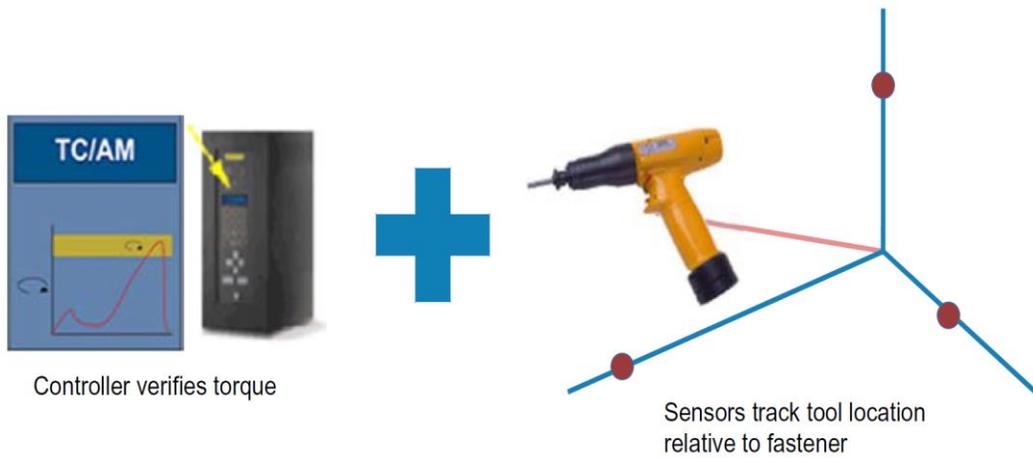


Figure 5. Tool tracking using an IR system (adapted from Ref. 5).

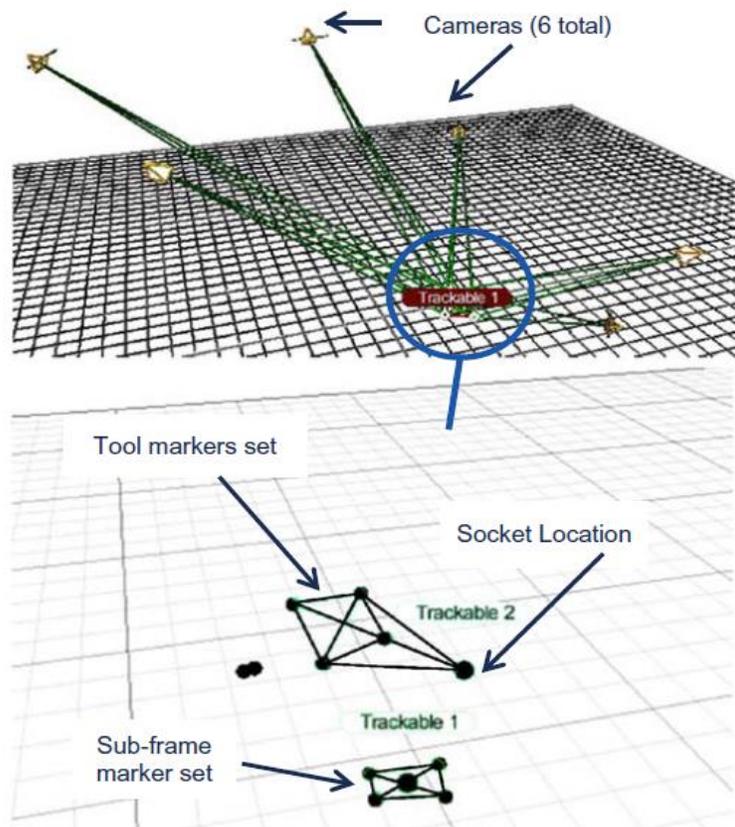


Figure 6. Screen shots from the Opti-track Tracking Tool Software (adapted from Ref. 5).

2.3 Vision system

The AI Vision Builder software-based vision inspection system from National Instruments (Austin, TX) has been investigated for bolt sequencing in the past [7]. Vision builder software is versatile and suitable for experimental study objectives. It has two algorithms to recognize a feature in an image: match pattern and geometric matching. The match pattern performs a bit-wise greyscale comparison of the raster images, while the geometric matching algorithm preprocesses the image to detect edges due to contrast and compares the detected edges to a reference geometric pattern.

Rusli and Luscher [7] found the match pattern to be faster and more accurate compared to geometric matching. Geometric matching is slower due to complex computations involved while recognizing edges. Additionally, geometric matching is more prone to lighting variations, as lighting affects the recognized edges significantly. In the match pattern process, while identifying a unique characteristic related to each bolt, the following criteria need to be considered:

- Features should be non-symmetrical.
- Features must have good contrast regions within the image.
- Features should be less sensitive to varying lighting conditions or shadows.
- Features must have blind holes to avoid inconsistent images in the background.

As can be seen in Fig. 7, the operator has to identify unique features associated with each bolt in order to satisfy the aforementioned requirements. Identifying these features requires vision system technical expertise; therefore, a shop operator might not be able to train the model with ease and will not be able to solve the issue, if any issue comes up. In addition, it is possible that such geometric patterns might not be unique for each bolt; therefore, this software will not be very useful in that case. Finally, the cost of this software

is \$2151[26], which can be too expensive to be implemented on every station in the assembly shop.

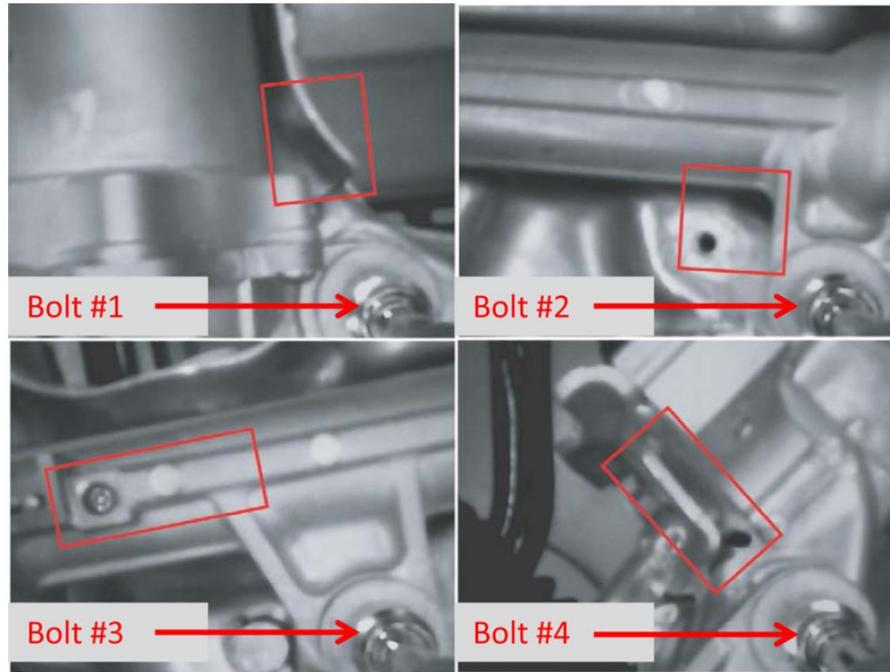


Figure 7. Feature pattern recognition associated with each fastener (steering rack) (adapted from Ref. 7).

Chapter 3: Deep Learning and Software Development

This chapter describes the convolutional neural network (CNN), advantages of CNN over standard neural network and different layers used in CNN. It also provides details on supervised machine learning and how supervised machine learning has been used in development of software. Furthermore, it also defines requirements of software and provides details on design and development of solution.

3.1 Convolutional Neural Network (CNN)

A neural network is an interconnection of artificial neurons that can exchange information between each other. Each connection has a numeric weight that is tuned during the training process by inputting a known dataset, so that the trained network is able to recognize a pattern or an image correctly. The network has multitude layers of feature-detecting neurons. The neurons in each layer respond to different combinations of inputs from the previous layers. As shown in Fig. 7, the layers are developed in such a way that the first layer detects primitive features in the input, and the second layer detects patterns in the patterns. CNNs typically deploy 5 to 25 distinct layers of pattern recognition.

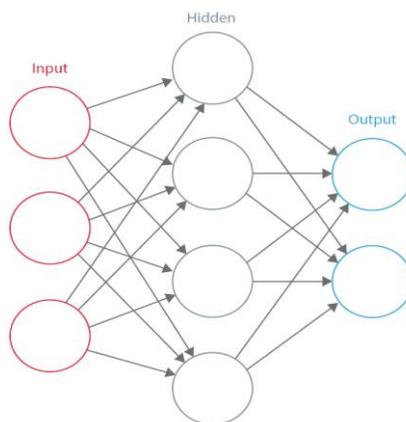


Figure 8. Neural Network (adapted from Ref. 12).

The inspiration behind neural networks is biological neural systems. There are numerous basic computational units in the brain, known as nerve cells or neurons. The neurons are connected to each other with synapses. The comparison between a biological neuron with a basic mathematical model is shown in Fig. 9.

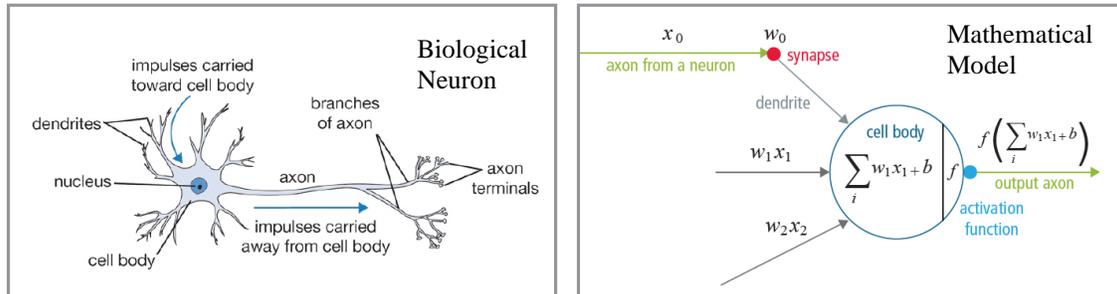


Figure 9. Biological neuron (left) & its mathematical model (right) (adapted from Ref. 8).

The input to a neuron is received by dendrites and the output signal produced flows along its axon. The branches of the axon connect to dendrites of other neurons via synapses. When the combination of input signals among its dendrites reaches a threshold value, the neuron is triggered, and the activation is communicated to successive neurons.

In the computational model, the signal x_0 travels along the axon of a neuron and interacts multiplicatively with dendrites of consecutive neurons to form a signal (w_0x_0) , where w_0 is the synaptic strength of the synapse. Synaptic strengths are tunable based on the interaction of one neuron with another. In the cell body, all the weighted signals are summed to get the final sum. If the final sum is above a defined threshold, the neuron is activated to send the signal along its axon to the consecutive neurons. The assumption in the computational model is that the precise timings of activation do not matter and only the frequency communicates information. To summarize, each neuron calculates the dot

product of inputs and weights, adds the bias to it, and applies non-linearity as a trigger function, such as a sigmoid function.

A Convolutional Neural Network (CNN) is a special case of neural network. A CNN has one or more convolutional layers with subsampling layers, which are followed by one or more fully connected layers as in a standard neural network. The visual mechanism, i.e., the visual cortex in the brain, laid down the foundation for design of CNNs. The cells in the visual cortex detect light in tiny, overlapping sub-regions of the visual fields. These visual fields are called receptive fields. The cells in the visual cortex act as local filters over the input space, and the more intricate cells have larger receptive fields. The convolution layer in a CNN performs the same function that is performed by the cells in the visual cortex [9]. A typical CNN for recognizing traffic signs is shown in Fig. 10. A set of features located in a small vicinity in a layer, known as local receptive field, is given as an input to the next layer. Features such as corners, endpoints, orientations, etc. can be extracted by local receptive fields. In the traditional model of pattern or image recognition, a custom-designed feature extractor gathers relevant information from the input and eliminates irrelevant variabilities. The extractor is followed by a trainable classifier, a standard neural network that classifies feature vectors into classes.

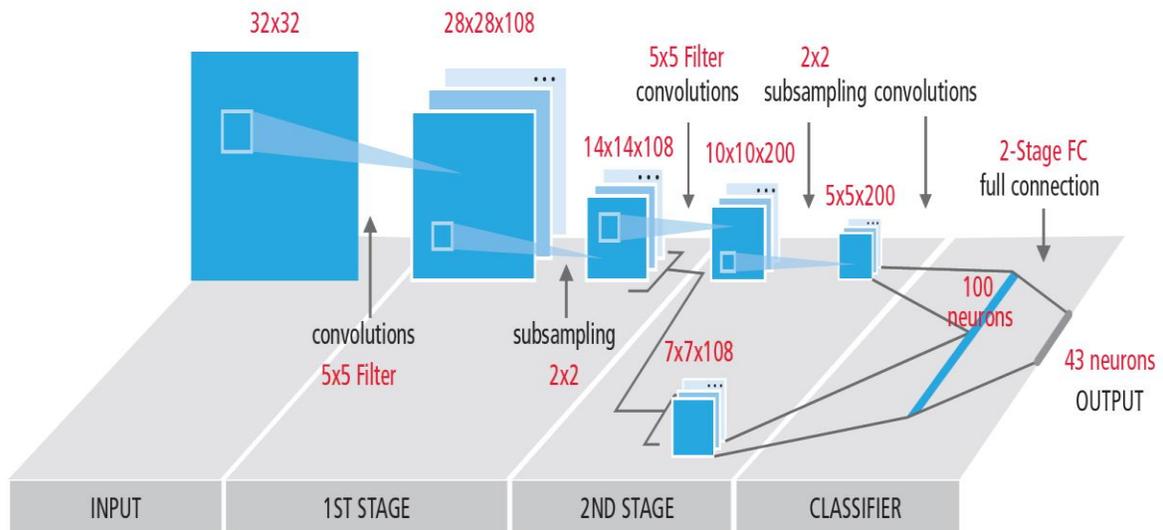


Figure 10. Typical Block Diagram of CNN (adapted from Ref. 10).

In a CNN, feature extraction is done by convolution layers, but the process is not custom designed. Convolution filter kernel weights are calculated during the training process. CNNs are useful in speech recognition, natural language processing, video analysis, and image and pattern recognition.

3.2 Advantages of CNN over other methods

The advantages of using a CNN are as follows:

- Training is easier and faster:** In a standard neural network, the number of training parameters is much higher; therefore, time taken to train the network also increases proportionally. In a convolutional neural network, the number of parameters is much lower; therefore, the training time is

reduced. In the case of a standard neural network, as there are more parameters., the probability of the addition of noise during the training process increases.

- **Robust to shifts and distortions in the image:** Detection of an image using a CNN is resilient to distortions introduced due to lighting conditions, different poses, change in shape due to camera lens, horizontal and vertical shifts, rotations, occlusions, etc.
- **Lower memory requirement:** With CNN, the same coefficients are used at different locations in space, so the memory requirement is drastically reduced.

3.3 Layers of CNN

3.3.1 Convolution layers

The convolution layers are used to extract different characteristics of the input. Low-level features, such as corners, edges and lines are extracted by the first convolution layer and the higher-level layers extract high-level features, such as objects and shapes in the image. The process of 3D convolution used in CNNs can be seen in Fig. 11. The dimensions of input are $N \times N \times D$ and the input is convoluted with H kernels, each having dimensions $k \times k \times D$. The convolution with each kernel produces one output feature, thus H features are produced using H kernels. The kernel is moved one element at a time from left to right, starting from top-left corner of input, and the kernel is moved one element in downward direction, once the kernel has reached the right end of input. This process is continued until the bottom-right corner is reached. For example, when $N = 32$ and $k = 5$,

there are 28 unique positions from left to right and 28 unique positions from top to bottom that the kernel can take. Corresponding to each of these positions, each feature in the output will contain 28×28 (i.e., $(N-k+1) \times (N-k+1)$) elements. In the sliding window process, $k \times k \times D$ elements of input and $k \times k \times D$ elements of kernel are multiplied element by element and accumulated for each kernel position.

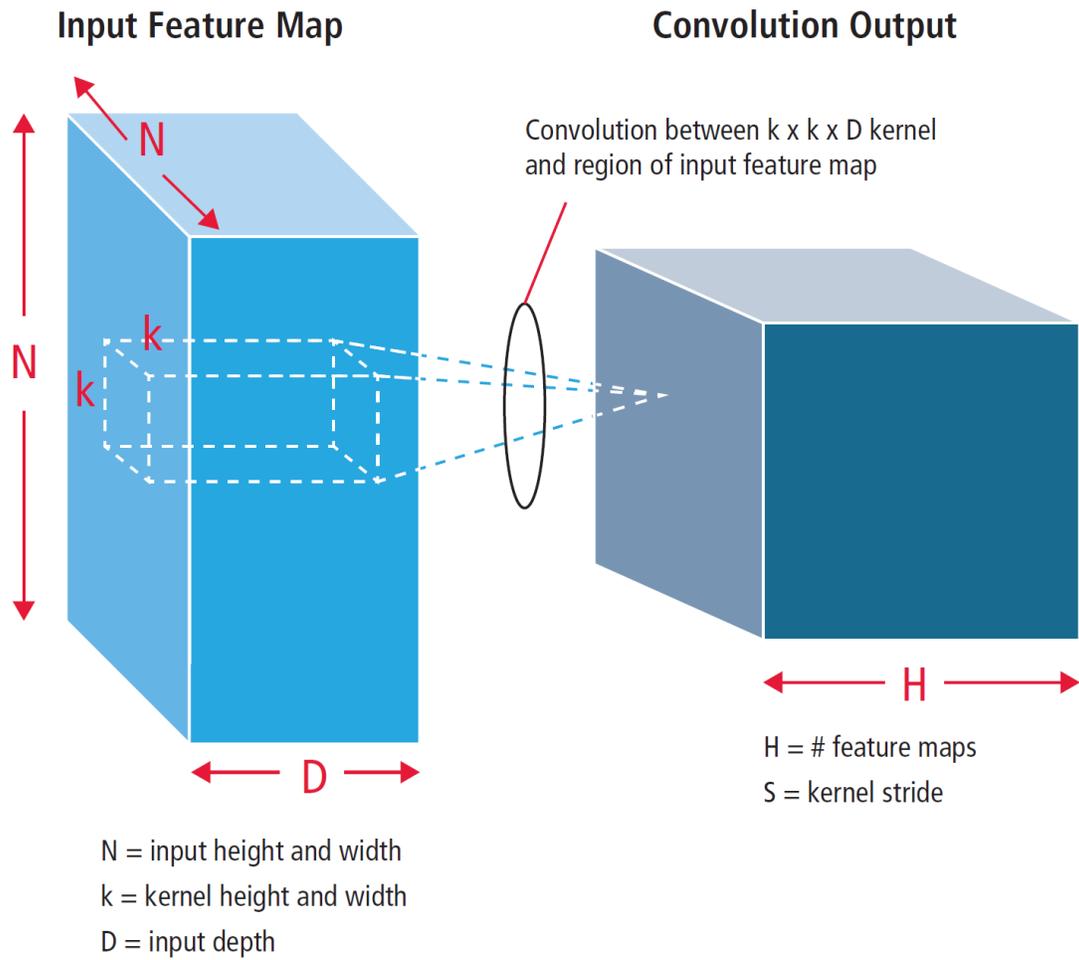


Figure 11. Pictorial representation of convolution process (adapted from Ref. 11).

3.3.2 Pooling/subsampling layers

The pooling layers reduce the resolution of the features, thus making them robust against distortion and noise. Pooling can be achieved in two ways: maximum pooling and average pooling. The input is divided into non-overlapping two-dimensional spaces in both cases. For example, in Fig. 12, layer 2 is the pooling layer. Each input feature is 28x28 and is divided into 14x14 regions of size 2x2. The average of four values in the region is calculated in case of average pooling; whereas, in the case of maximum pooling, the maximum value out of four values is selected. Input to pooling layers has dimensions of 4x4. The image is divided into 4 non-overlapping matrices of dimensions 2 x 2 in the case of 2 x 2 subsampling. For averaging, fractions are rounded to the nearest integer.

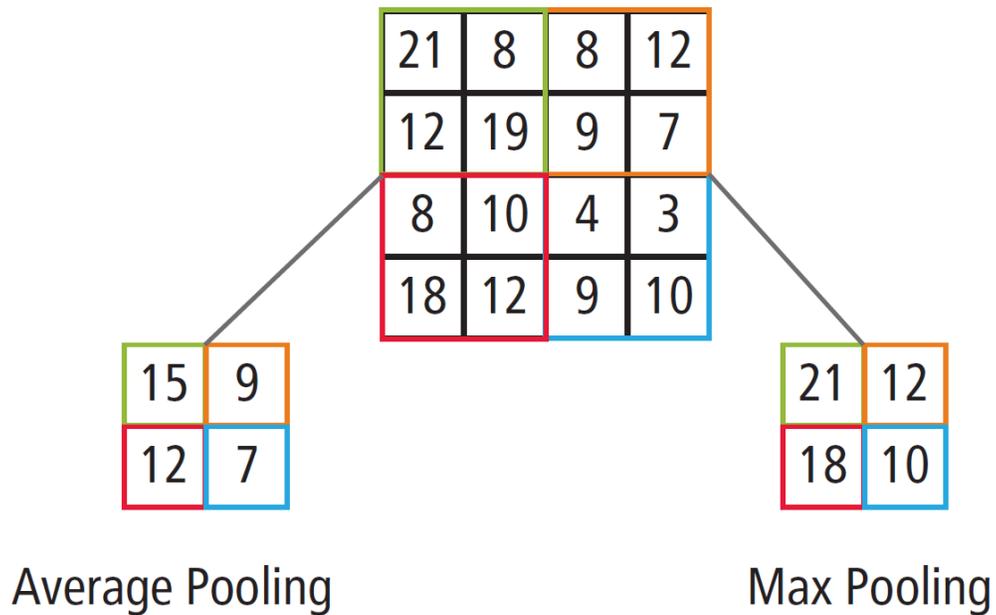


Figure 12. Pictorial representation of maximum & average pooling (adapted from Ref.

3.3.3 Non-linear layers

CNNs use non-linear trigger functions to signal distinct identification of likely features on each hidden layer. CNNs use continuous trigger (non-linear) functions and specific functions, such as rectified linear (ReLU) functions, to implement non-linear triggering efficiently.

a. ReLU

A ReLU implements the function $y = \max(x,0)$, so the input and output sizes of this layer are the same. It increases the nonlinear properties of the decision function and of the overall network without affecting the receptive fields of the convolution layer. The training rate is much faster with ReLU as compared to other non-linear functions, such as hyperbolic tangent, sigmoid, etc. ReLU functionality is illustrated in Fig. 13, with its transfer function plotted above the arrow.

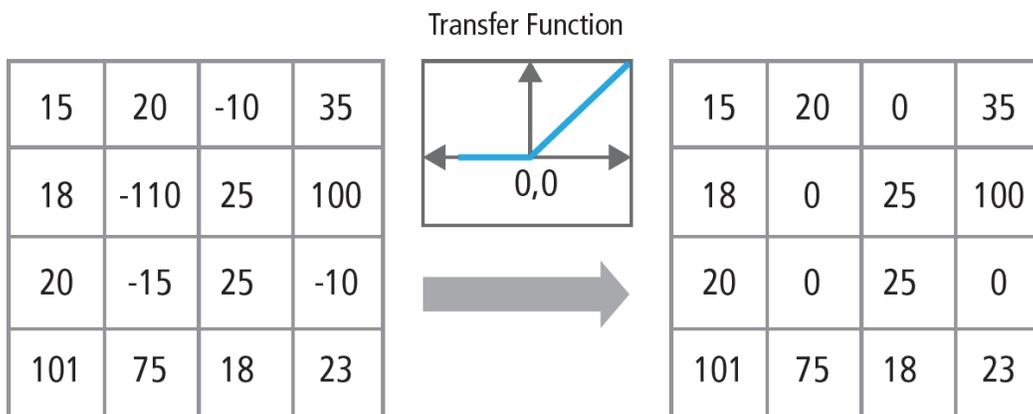


Figure 13. Pictorial representation of ReLU functionality (adapted from Ref. 13).

b. Continuous trigger (non-linear) function

With the continuous trigger function, the non-linear layer operates on every element in each feature. A continuous trigger function can be a sigmoid, a hyperbolic tangent or the absolute value of the hyperbolic tangent. Fig. 14 demonstrates how non-linearity is applied on each element.

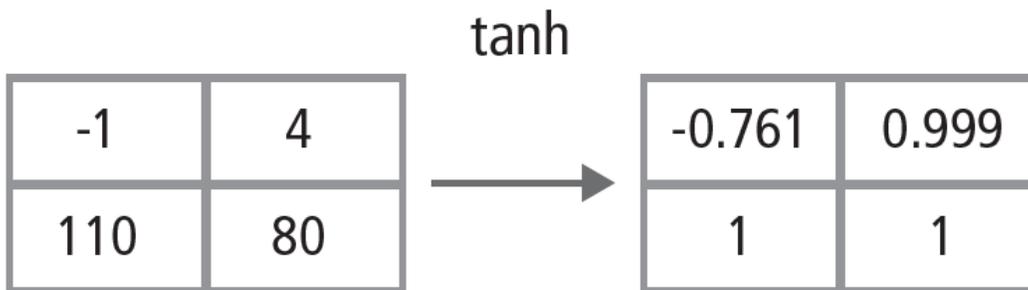


Figure 14. Pictorial representation of ReLU functionality (adapted from Ref. 13).

3.3.4 Fully connected layers

Fully connected layers are the final layers of a CNN. They mathematically sum the weights of the features of the previous layer. In the case of a fully connected layer, the elements of all the features of the previous layer are used to calculate the element of each output feature. Fig. 15 explains the fully connected layer L . Layer $L-1$ has two features, each of which is 2×2 , i.e., it has four elements. Layer L has two features, each having a single element.

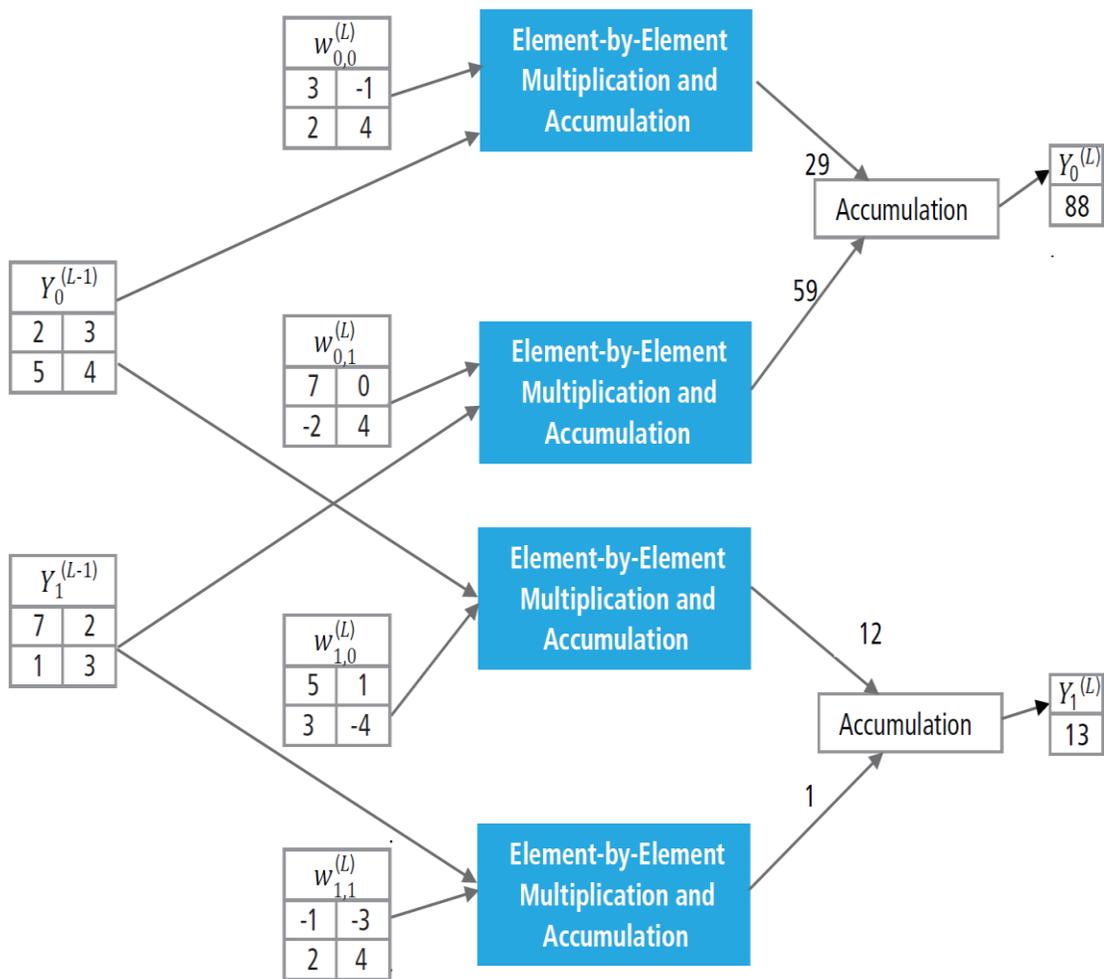


Figure 15. Pictorial representation of ReLU functionality (adapted from Ref. 13).

3.4 Supervised Machine Learning and Software Development

In supervised machine learning, a dataset of inputs and associated output labels is given as a training set and this data is used to infer a function. This function is used to determine the class labels for unseen instances. The following are the main steps in the supervised machine learning model:

- Model construction
- Model training
- Model testing
- Model evaluation

Model Construction: There are many supervised machine learning algorithms, such as logistic regression, linear regression, support vector machines and neural networks, that can be used to construct the model, based on the input dataset and complexity of the problem. In the case of bolt crosstalk in the present work, convolutional neural networks (CNN) were used. To implement the CNN algorithm for this solution, Python has been used as the programming language and the open-source neural network library, Keras, which is based on TensorFlow, has been used to construct the model. TensorFlow is an open source library developed by Google that is mainly used to perform numerical operations to model Deep Learning models. The sequential CNN model is a linear stack of layers. The construction of the model begins by defining an object as ‘`model=Sequential()`’, then a number of layers are added using ‘`model.add(type_of_layer())`’, depending on the complexity of the problem. After adding the required number of layers, the model is compiled using the Python code: `model.compile(loss='name_of_loss_function', optimizer='name_of_optm_algo')`. Then Keras communicates with TensorFlow to construct the model. Two important aspects to be considered while compiling the model are: the loss function and optimizer algorithms. The loss function shows the accuracy of

each prediction made by the model and the optimizer algorithm is used to tune the weights associated with neurons in each layer.

Model Training: After creating the CNN model, the model is trained using ‘model.fit(training_data, expected_output)’. The accuracy of training is reported at the end of training process. Once the model has been trained, the model can be saved by ‘model.save(“name_of_model”)’.

Model Testing: Once the model has been trained and saved, data that has not been used during training is used to test the accuracy of the model.

Model Evaluation: If the accuracy from the model testing phase is in line with the required accuracy, then the model can be used for evaluation of new data. If not, the model can be trained again with a larger training dataset.

3.5 Software Requirements

Requirements for the software are as follows:

- **User-friendly Graphical User Interface (GUI):** Operators with basic computer knowledge should be able to operate the system.
- **Quick Detection:** The software must identify the bolt in shortest possible time so that the cycle time of the station is not affected by implementing this system.
- **Accurate Detection:** All the bolts on a station should be identified accurately to minimize re-work.
- **Current Status:** The software needs to show the number of products for which the operator has not followed the tightening sequence.

- **Interface with DC tool controller:** DC tool controller is used to control the torque applied by DC tool motor and give torque tightening details to the programmable logic controller (PLC). If the camera has been assembled on a DC tool, the software should interface with the DC tool controller so that the torque value can be mapped with each bolt.
- **Connection with server:** The software must receive model specifications from a server and should send the data to the server after tightening has been completed.

The interface with the DC tool controller requires knowledge of the specific company's controller. The main goal of the present work was to develop a universal solution that can interface with all DC tool manufacturers. Therefore, the focus has been on requirements 1, 2 and 3.

3.6 Software Details

A graphical user interface has been developed for the software using the Tkinter library in Python. When the user opens the software, the screen in Fig. 16 will pop up.

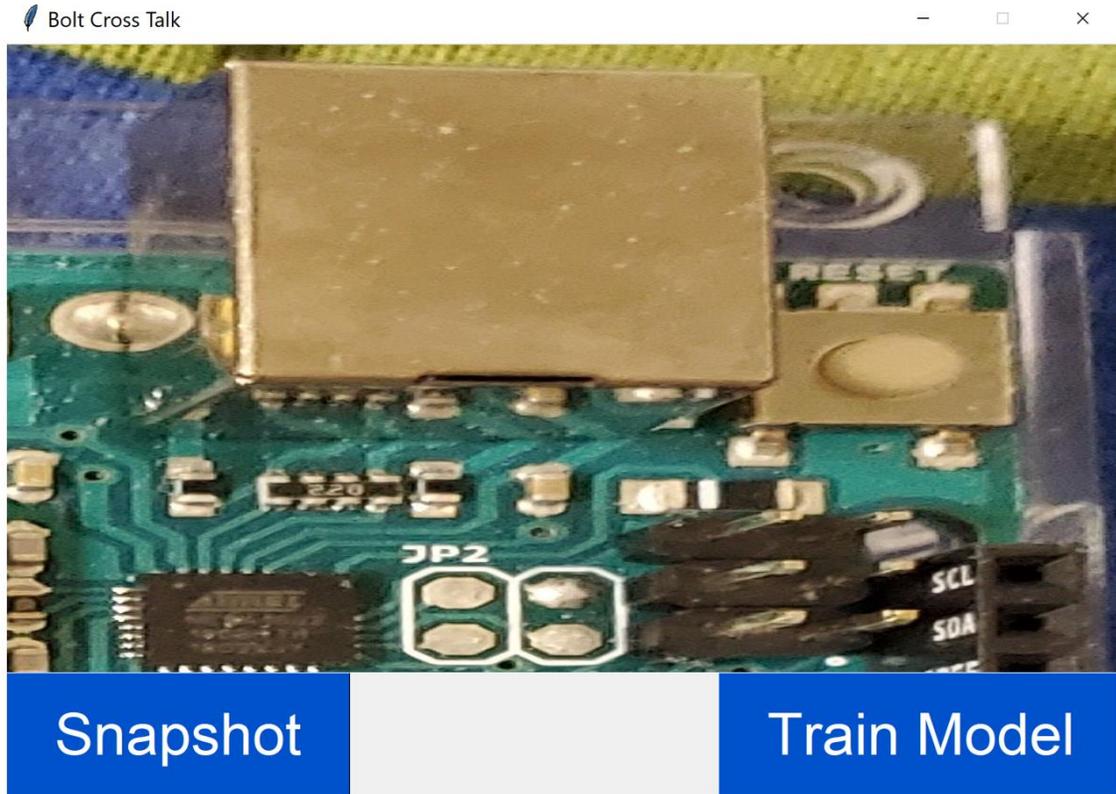


Figure 16. Main screen of GUI of software.

The main functional components of the program are as follows: :

- 1. Camera Display:** Shows the view of the camera mounted on the tool to the operator.
- 2. Snapshot/Push Button:** When the operator goes to the bolt tightening position, he presses the button on the tool or clicks the Snapshot button in the software to click the picture. The image is pre-processed and supplied as an input to the trained model. The trained model writes the output to a csv file.

- 3. Train Model:** Train Model button is used to train the model for a given problem.

3.6.1 Camera Display

A compact camera is mounted on the tool to capture images. The camera is set up such that a unique characteristic related to every bolt can be used to identify the bolt. In Fig. 17, it can be seen that for the position of bolt encircled in orange, the box in white is the unique feature. If this feature is detected, then the corresponding bolt position can be detected. This lays the foundation of the solution.

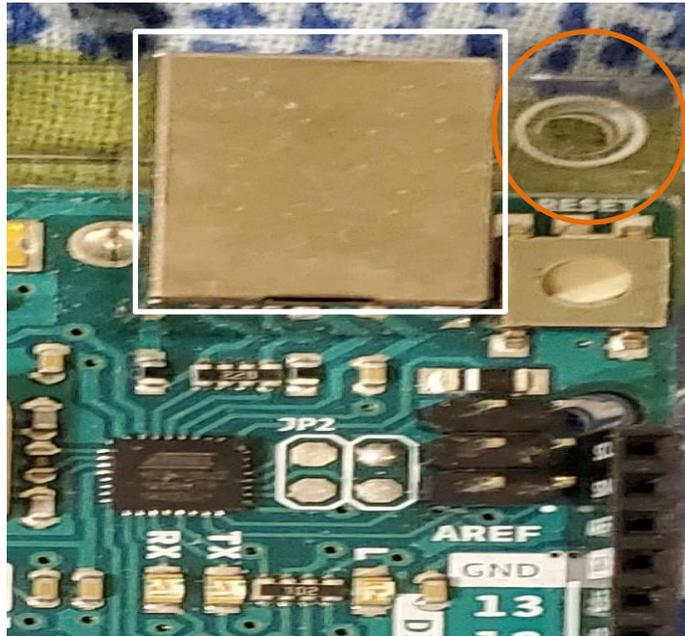


Figure 17. Unique characteristic related to encircled bolt.

3.6.2 Snapshot/Push Button

The following flowchart describes the functioning of the overall system when developed solution is deployed on a DC tool. When the unfinished product enters the station, a limit switch is actuated to fetch the data for that particular model. The operator

takes the DC tool with the camera mounted on it to the bolt position and presses the button. The button actuates image capture at that position and sends the image to the software for image processing. The trained model decides if the operator is at the correct position or informs the operator of incorrect position by lighting up red LED. Once the operator takes the tool to the correct position, then the 'OK' signal is communicated to DC tool. Then, the operator is able to start the tool to begin tightening. The same procedure is followed for all the bolts for a given station. In case the button stops working, the operator can click on 'Snapshot' to capture the image.

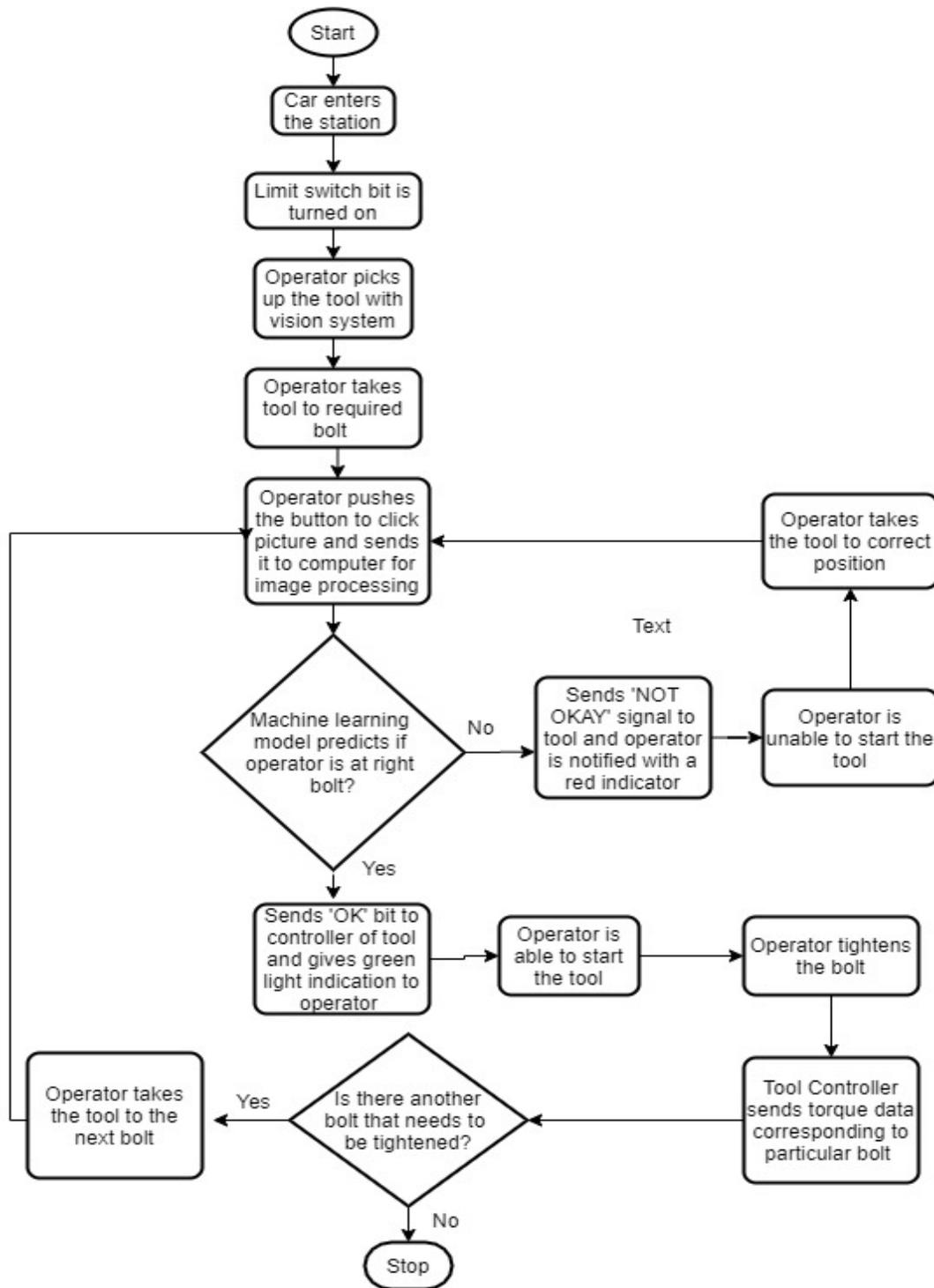


Figure 18. Flowchart of overall system.

The flowchart shown in Fig. 18 details the overall system when the developed system is integrated with a DC tool. As reading data from a file using a Programmable Logic Controller (PLC) and giving it as an input to DC tool controller is well-established in manufacturing industry, so the focus of this research is on the development of an algorithm and software that can write an output to a file and use well-established manufacturing practices to interface software with DC tools. The flowchart shown in Fig. 19 describes the process that is relevant to work done in this research.

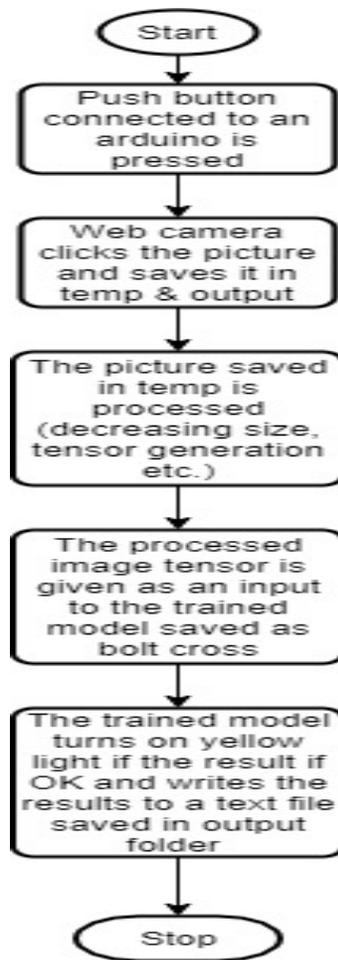
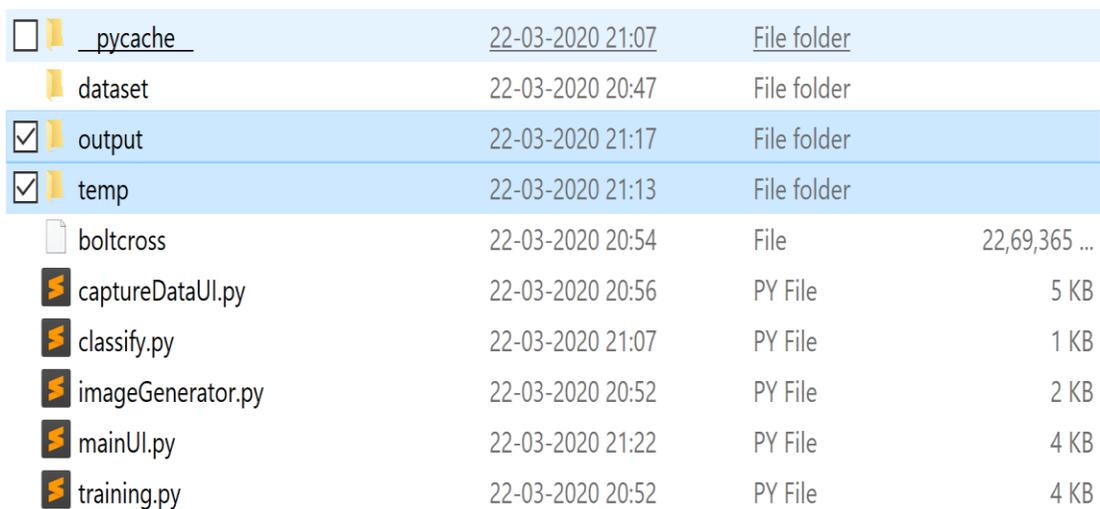


Figure 19. Flowchart of developed system.

Details of the steps defined in the flowchart (refer to Fig. 19) are as follows:

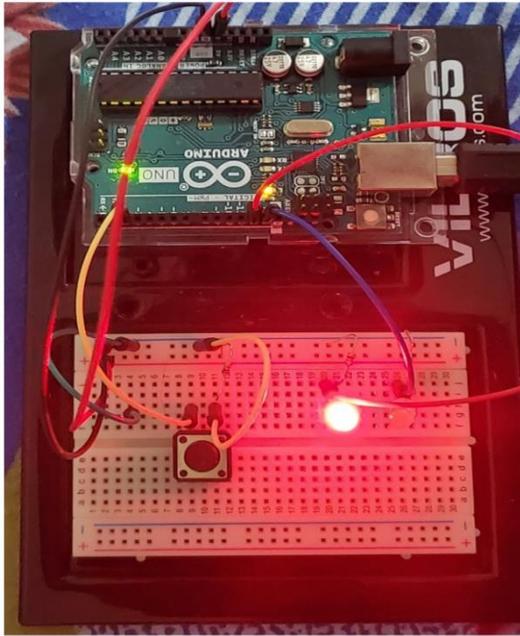
- Push button connected to Arduino is pressed to capture a test image.
- The captured image is saved in two folders: temp and output. All the captured images are stored in output folder so that the timestamp as image name can be used as a reference in case of malfunction of the software. The image saved in the temp folder is sent to the trained model ‘boltcross’ for evaluation.



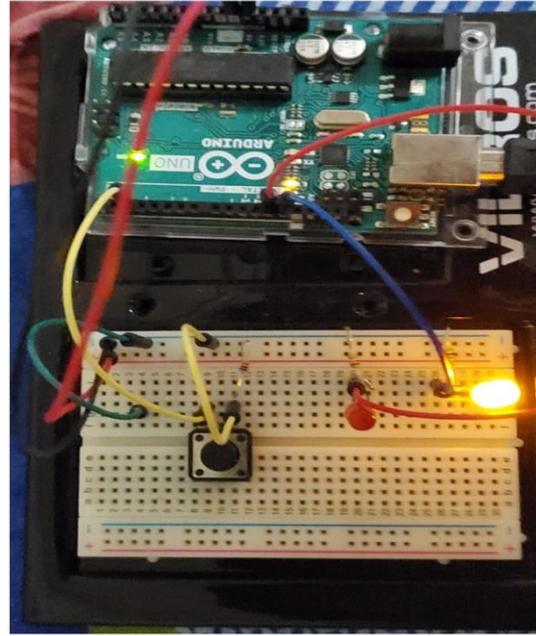
<input type="checkbox"/>	__pycache__	22-03-2020 21:07	File folder	
	dataset	22-03-2020 20:47	File folder	
<input checked="" type="checkbox"/>	output	22-03-2020 21:17	File folder	
<input checked="" type="checkbox"/>	temp	22-03-2020 21:13	File folder	
	boltcross	22-03-2020 20:54	File	22,69,365 ...
	captureDataUI.py	22-03-2020 20:56	PY File	5 KB
	classify.py	22-03-2020 21:07	PY File	1 KB
	imageGenerator.py	22-03-2020 20:52	PY File	2 KB
	mainUI.py	22-03-2020 21:22	PY File	4 KB
	training.py	22-03-2020 20:52	PY File	4 KB

Figure 20. Screenshot showing output and temp folders.

- If the result is OK, then the yellow LED turns on. If not, then the red LED turns on as shown in Fig. 21.



NOT OK



OK

Figure 21. LEDs on NOT OK and OK outputs.

- The test results are stored in a text file in the output folder as shown in Fig. 22.

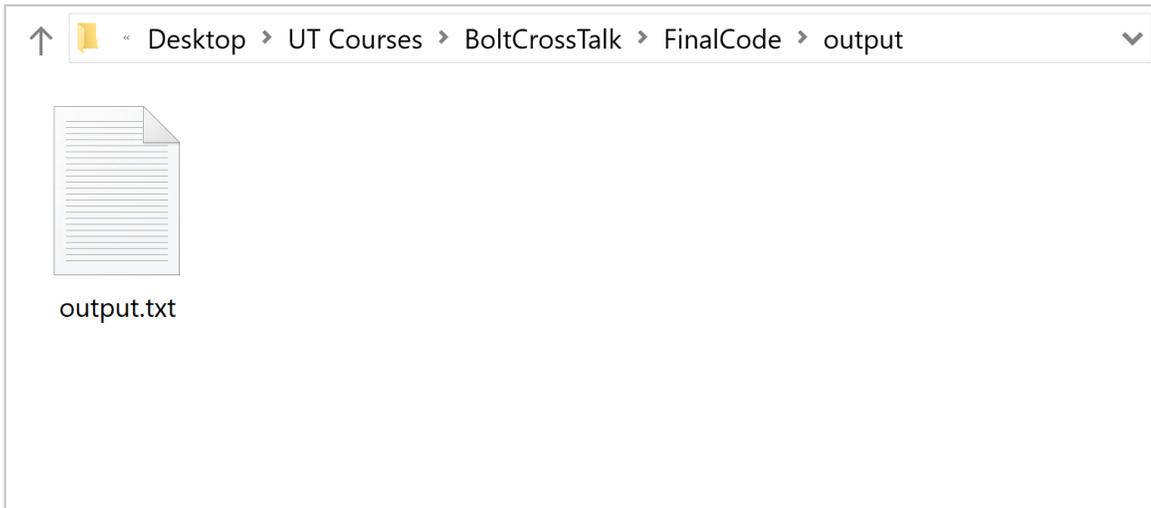


Figure 22. Result file in output folder.

3.6.3 Train Model

When the operator wants to train the model for the application, he clicks the ‘Train Model’ button. On clicking the Train Model button, the screen shown in Fig. 23 pops up.

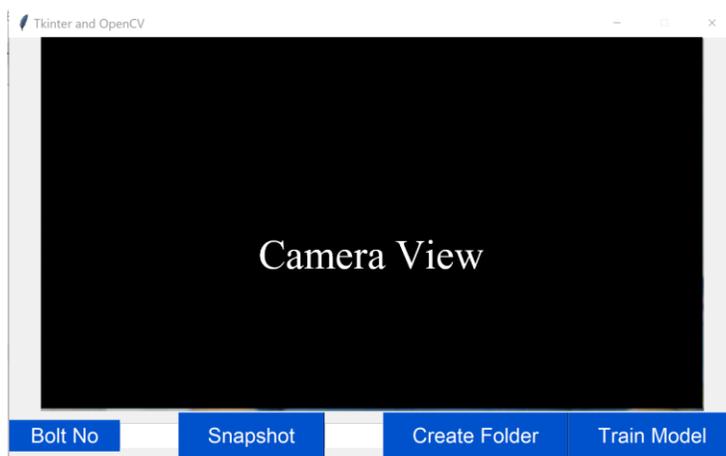


Figure 23. Train Model GUI.

The operator inputs the number of unique bolt positions for a station in the encircled text box as shown in Fig. 24.

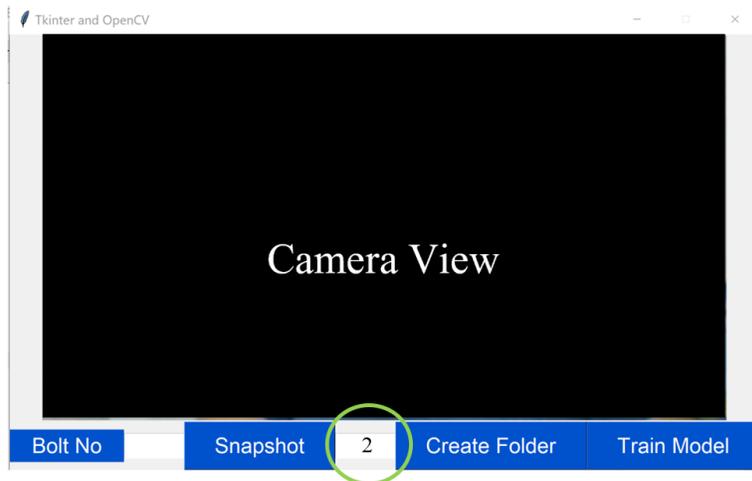


Figure 24. Number of unique bolt positions.

When the operator clicks 'Create Folder' button, the number of folders corresponding to number of unique positions is created in the dataset folder, as shown in Fig. 25.

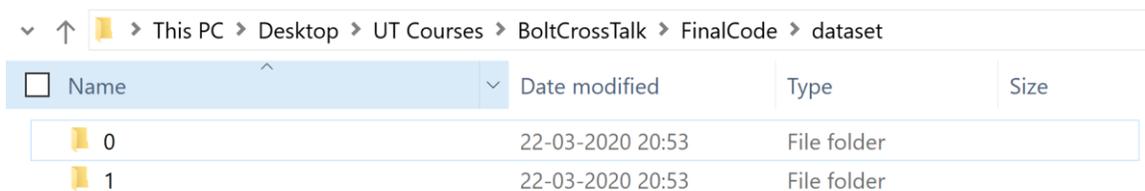


Figure 25. Folders created in dataset.

To collect data required for training the model, the user inputs the bolt position for which the pictures are being captured in the encircled text box shown in Fig. 26.

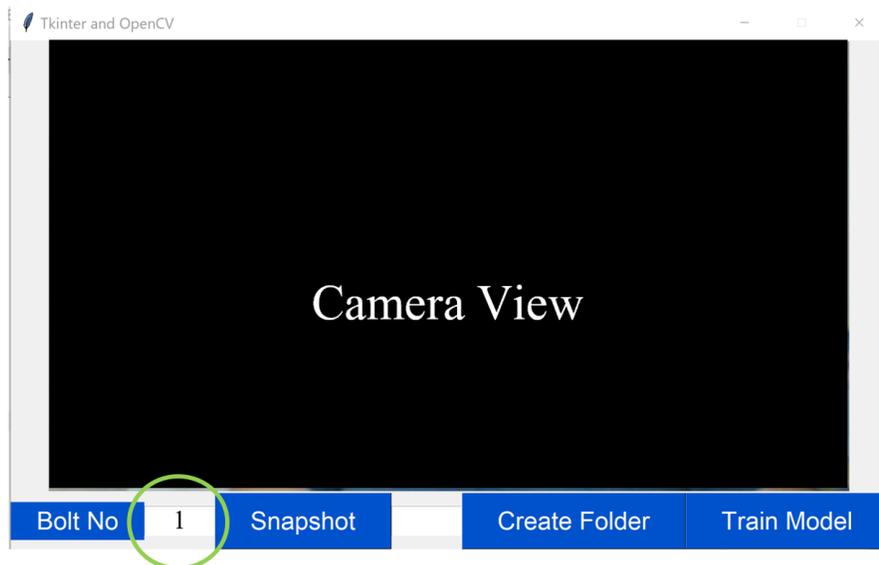


Figure 26. Data to be stored in given folder.

Then the user clicks the 'Snapshot' button to save the image in the given folder. The folder name is $n-1$, where n is the number given as an input by the user, as shown in Fig. 27.

↓ ↑ > This PC > Desktop > UT Courses > BoltCrossTalk > FinalCode > dataset > 0



Figure 27. Clicked picture in given folder number.

When the user has created a dataset for all the bolt tightening positions, the user clicks the ‘Train Model’ button. As the number of images that can be clicked for a position is limited, image transformations must be applied in order to increase the dataset for each position. Using Keras’ ‘ImageGenerator’ class, it is possible to apply random transformations to increase the dataset. The method ImageDataGenerator of the ImageGenerator class has the following arguments:

1. **rotation_range:** This parameter specifies rotations of images. The input parameter is an angle in degrees, ranging from 0 to 180.
2. **width_shift_range:** This parameter specifies horizontal shifts in the images.
3. **height_shift_range:** This parameter specifies vertical shifts in the images.
4. **shear_range:** This parameter displaces each pixel in a fixed direction.
5. **zoom_range:** This parameter specifies random zooming of parts of image.
6. **horizontal_flip:** This parameter randomly flips inputs horizontally.
7. **vertical_flip:** This parameter randomly flips inputs vertically.

8. **fill_mode**: This parameter indicates the method of filling newly formed pixels.

There are more arguments that can be used by ImageDataGenerator class. The following code is used to read the images from all folders and apply transformations to them:

```
datagen = ImageDataGenerator(
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')
folder=os.path.join(os.getcwd(),"dataset")
subfolders = [ f.path for f in os.scandir(folder) if f.is_dir()]
print(subfolders)
for subfolder in subfolders:
    subfiles = [ f.path for f in os.scandir(subfolder) if f.is_file()]
    for fil in subfiles:
        img = load_img(fil)
        x = img_to_array(img)
        x = x.reshape((1,) + x.shape)
        x.shape
        i = 0
        for batch in datagen.flow(x, batch_size=1,
            save_to_dir=subfolder, save_prefix='e1', save_format='jpeg'):
            i += 1
            if i > 50:
                break
        break
```

Figure 28. Python code to apply transformations to dataset.

Fig. 29 shows the images that have been captured by the vision system Since the data is limited, random image transformations need to be applied to increase the dataset. Fig. 30 shows the dataset for one of the bolt locations, when the code for random image transformations has been executed.



Figure 29. Before random transformations of dataset.

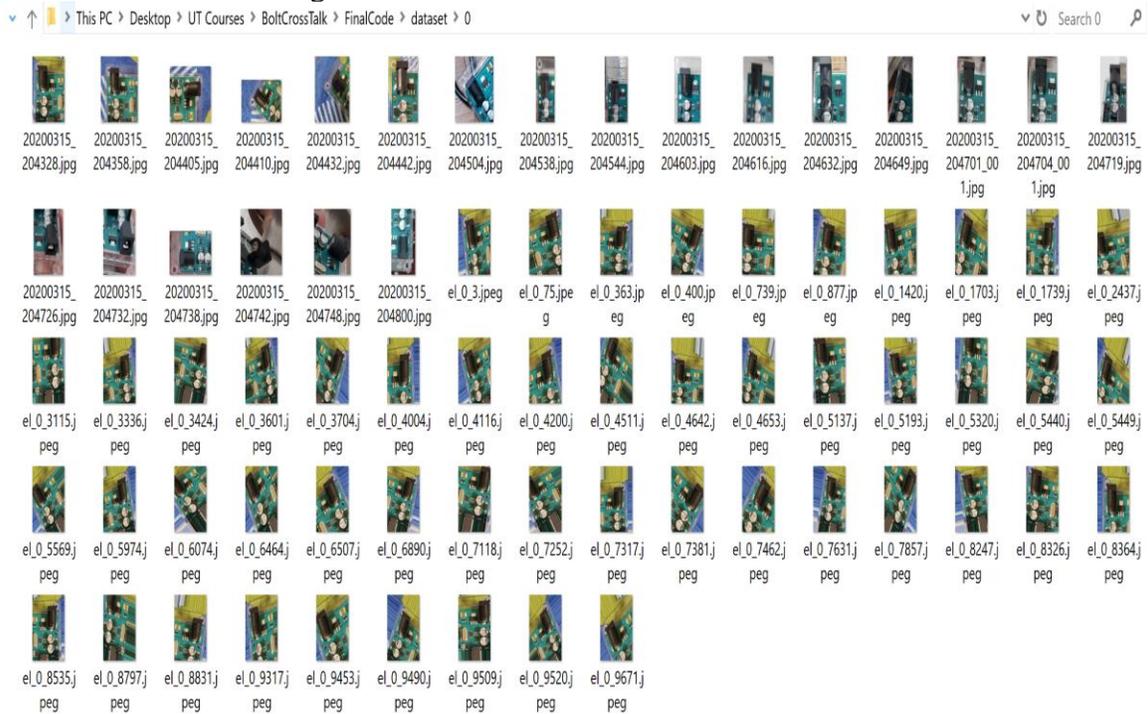


Figure 30. Dataset after image transformations.

After the dataset has been extrapolated, the code below is used to create a tensor of images using the Keras library. The dimensions of the tensor of an image are height of image * width of image * 3. '3' signifies RGB values of pixel. The names of folders are used as labels.

```
IMAGE_DIMS = (50, 50, 3)
print("[INFO] loading images...")
imagePaths = sorted(list(paths.list_images("dataset")))
random.seed(2)
random.shuffle(imagePaths)
# initialize the data and labels
data = []
labels = []
# loop over the input images
for imagePath in imagePaths:
    # load the image, pre-process it, and store it in the data list
    try:
        image = cv2.imread(imagePath)
        image = cv2.resize(image, (IMAGE_DIMS[1], IMAGE_DIMS[0]))
        image = img_to_array(image)
        data.append(image)
        l = imagePath.split(os.path.sep)[-2].split("_")
        labels.append(l)
    except:
        pass
data = np.array(data, dtype="float") / 255.0
labels = np.array(labels)
(x_train, x_test, y_train, y_test) = train_test_split(data,
    labels, test_size=0.15, random_state=2)
```

Figure 31. Python code to create tensor of images.

Once all the images have been converted into tensors with their respective labels, a CNN is constructed using the Keras library. The model used in this research consists of three groups of layers, where the convolution layers (Conv 2D) in conjunction with non-

linear layers (ReLU) and pooling layers (Max Pooling 2D) are followed by two tightly bound layers (Dense).

```
model=Sequential()  
model.add(Conv2D(50,(3,3),activation='relu',input_shape=(50,50,3)))  
model.add(MaxPooling2D(pool_size=(2,2),strides=1))  
model.add(Conv2D(100,(3,3),activation='relu'))  
model.add(MaxPooling2D(pool_size=(2,2),strides=1))  
model.add(Flatten())  
model.add(Dense(1000,activation='relu'))  
model.add(Dense(2,activation='softmax'))
```

Figure 32. Python code to construct CNN.

The parameter 50 in Conv2D indicates the number of output filters in the convolution. The pair (3,3) denotes the kernel size and determines the width and height of the 2D convolution window. Another important component is the input shape, which is an input array of pixels. Other convolution layers have been constructed in a similar way, but they do not include the input shape. The activation function that has been used is rectified linear function (ReLU) (refer to Fig. 33). Using zero as a threshold, this function cuts off unnecessary details in the channel if $x < 0$ and the volume of array of pixels remains the same if $x > 0$.

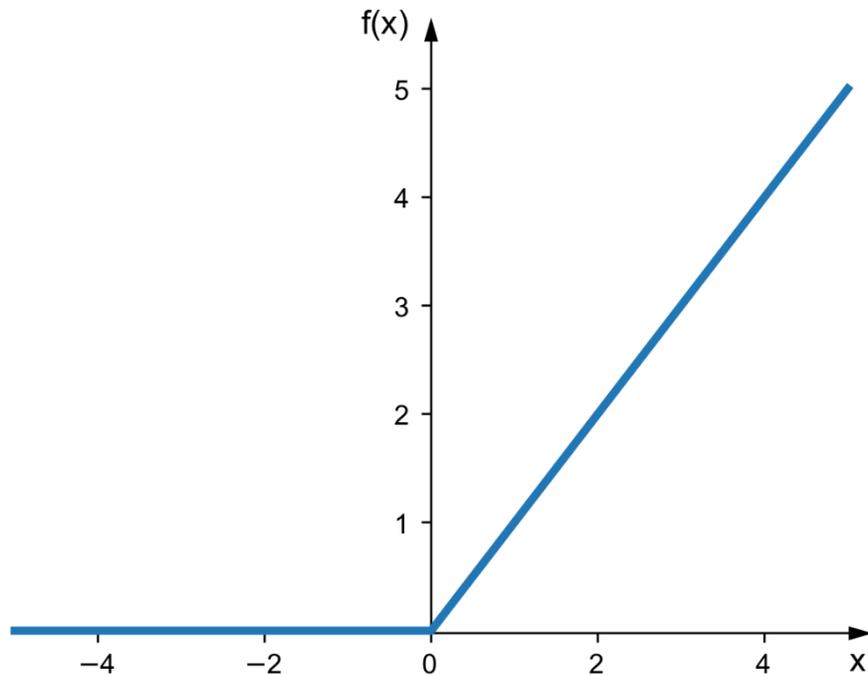


Figure 33. Rectified Linear (ReLU) function.

The pooling operation for spatial data is performed by the Max Pooling 2D layer. The pair (2, 2) is the pool size, which halves the input in both directions. The flatten function converts the tensor into a 1-D tensor. The dense layer is the implementation of the equation: $\text{output} = \text{activation}(\text{dot}(\text{input}, \text{kernel}) + \text{bias})$. This equation denotes the dot product between the input tensor and the weight kernel matrix defined in the dense layer. After taking the dot product, the bias vector is added, and element-wise activation of output values is taken.

The final step in training the model is compilation of the model and saving it. If the number of positions is more than 2, the categorical cross entropy loss function can be used. In the case of 2 positions, the binary cross entropy loss function is used. The optimizer algorithm that has been used is 'adam', which is good for recurrent neural networks. The accuracy metrics shows the performance of the model.

```
optimizers.Adam(learning_rate=0.0001)
model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
```

Figure 34. Python code to compile the model.

Batch size is defined as the number of training examples in one forward/backward pass. One epoch is one forward pass and one backward pass over all the training examples. 'Validation split' is used to split the data into two parts: training and validation. Training data is the one using which tuning of weights corresponding to each neuron takes place and model is then tested for accuracy on validation set. The model is saved as 'boltcross'.

```
model.fit(x_train,y_train_one_hot,batch_size=25,epochs=5,
validation_split=0.20)
model.save("boltcross")
```

Figure 35. Python code to validate and save model.

The progress of training, including accuracy and loss of data for each epoch, can be seen in Fig. 36. The number of epochs is chosen such that maximum accuracy for validation as well as the training set can be achieved, but at the same time, overfitting of data can be avoided. Overfitting is the production of an analysis that corresponds too closely to a particular set of data, and therefore analysis may fail to predict future observations reliably. Increased accuracy of training data with each epoch can be observed in Fig. 37. If we increase the number of epochs further, a point is reached where the accuracy of the training set might keep increasing but the accuracy of the validation set decreases with each epoch (refer Fig. 38). This results from overfitting of data.

```

Train on 164 samples, validate on 42 samples
Epoch 1/8
164/164 [=====] - 31s 192ms/step - loss: 12.1394 - accuracy: 0.5000 -
val_loss: 0.6103 - val_accuracy: 0.8095
Epoch 2/8
164/164 [=====] - 31s 190ms/step - loss: 0.5663 - accuracy: 0.7744 -
val_loss: 0.6666 - val_accuracy: 0.6905
Epoch 3/8
164/164 [=====] - 37s 226ms/step - loss: 0.4224 - accuracy: 0.7866 -
val_loss: 0.3589 - val_accuracy: 0.8810
Epoch 4/8
164/164 [=====] - 31s 186ms/step - loss: 0.1950 - accuracy: 0.9512 -
val_loss: 0.2749 - val_accuracy: 0.9048
Epoch 5/8
164/164 [=====] - 34s 208ms/step - loss: 0.0575 - accuracy: 0.9756 -
val_loss: 0.3275 - val_accuracy: 0.9286
Epoch 6/8
164/164 [=====] - 32s 192ms/step - loss: 0.0264 - accuracy: 1.0000 -
val_loss: 0.1815 - val_accuracy: 0.9762
Epoch 7/8
164/164 [=====] - 27s 163ms/step - loss: 0.0072 - accuracy: 1.0000 -
val_loss: 0.1975 - val_accuracy: 0.9762
Epoch 8/8
164/164 [=====] - 30s 182ms/step - loss: 0.0016 - accuracy: 1.0000 -
val_loss: 0.1973 - val_accuracy: 0.9762
37/37 [=====] - 0s 10ms/step

```

Figure 36. Output of results after each epoch.

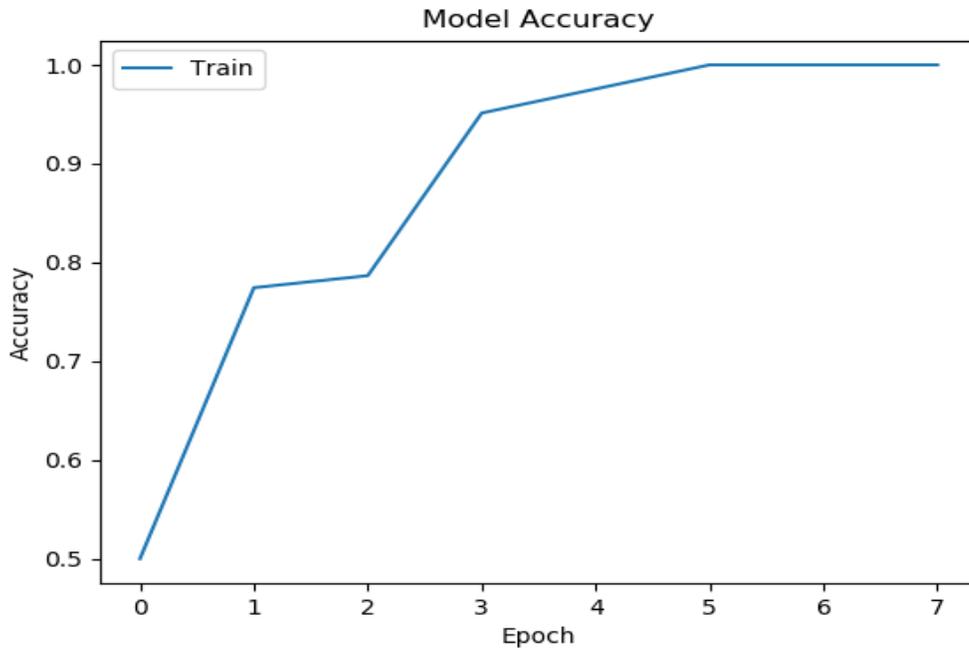


Figure 37. Plot showing increase in accuracy with each epoch.

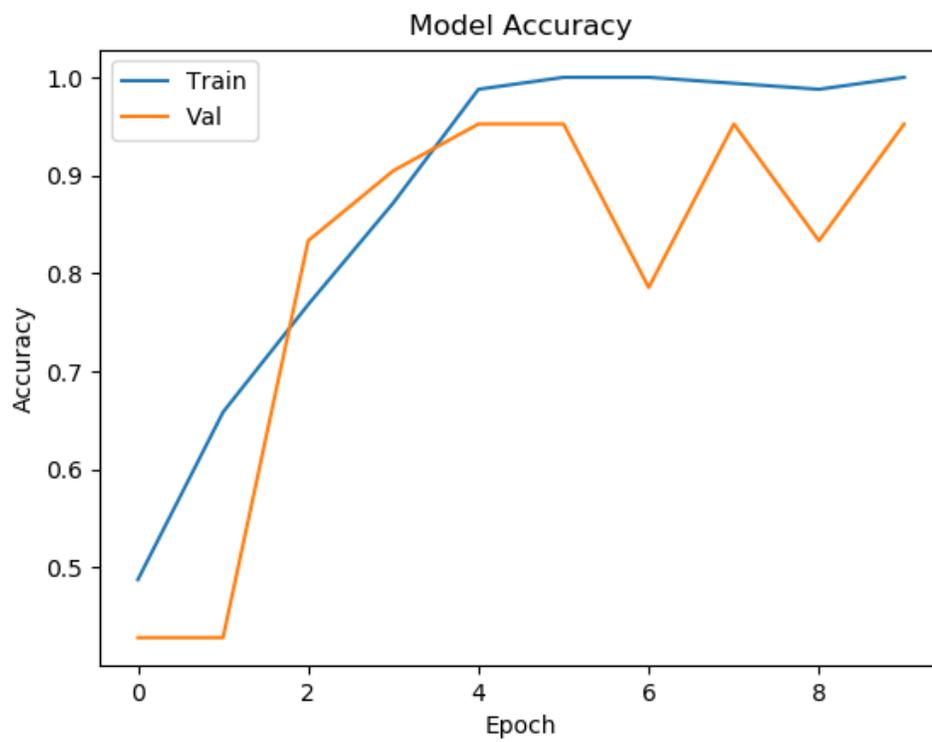


Figure 38. Plot showing overfitting of data.

Chapter 4: Testing and Results

This chapter provides the details of the testing of the software and studies the effect of brightness of image on probability of detecting a position.

4.1 Detailed description of software testing

An Arduino UNO was used to evaluate the performance of the software. The Arduino Uno's base has a number of holes that are used to fasten it to mounting surfaces. See Appendices A and B for specifications on the PC and Arduino used in this research. Each of the positions was assigned a number, as shown in Fig. 39. Arduino UNO was chosen because of following reasons:

- Availability
- Each position of hole has a unique characteristic associated with it (refer Fig. 40)
- At least 50% of the area of the image is green in color; therefore, it is possible that the algorithm might give false positives or false negatives.

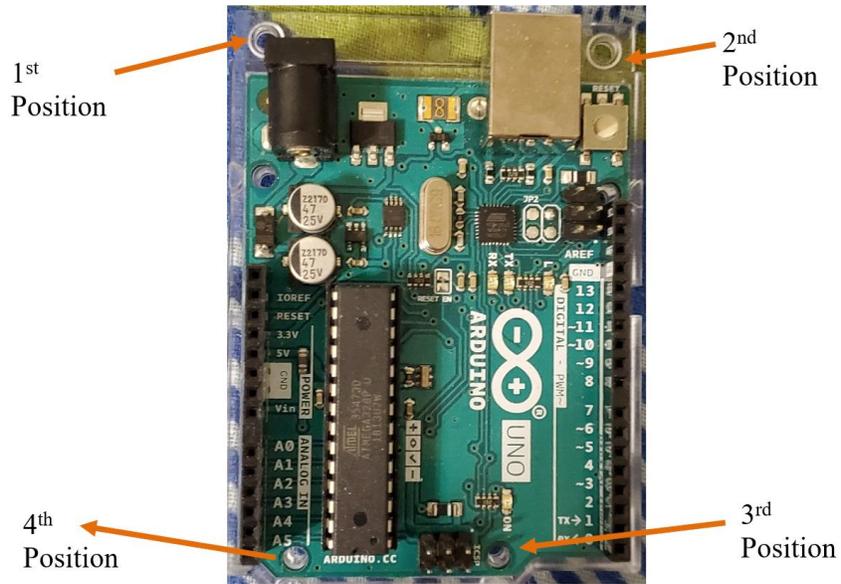


Figure 39. Arduino UNO with different bolt positions.

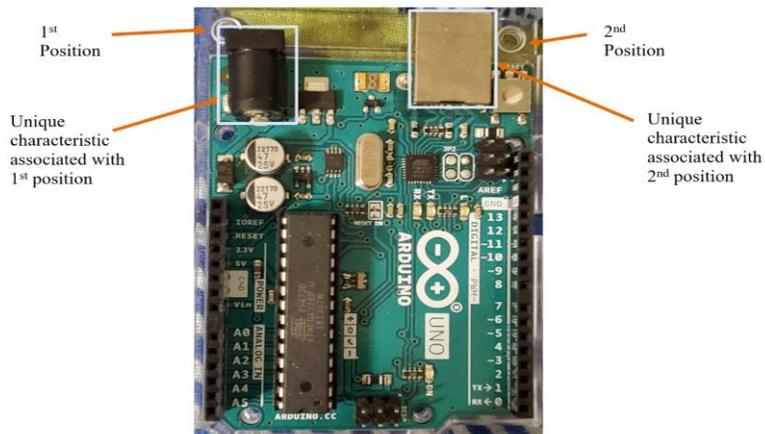


Figure 40. Examples of position and unique characteristics.

The code was tested for all four positions. The image for a particular position was captured using the button connected to the Arduino (Fig. 41), which signals the software using serial communication to capture the picture. The trained model, saved as ‘bolt crosstalk’, was used to classify the image and result was saved in .txt file in the output folder.

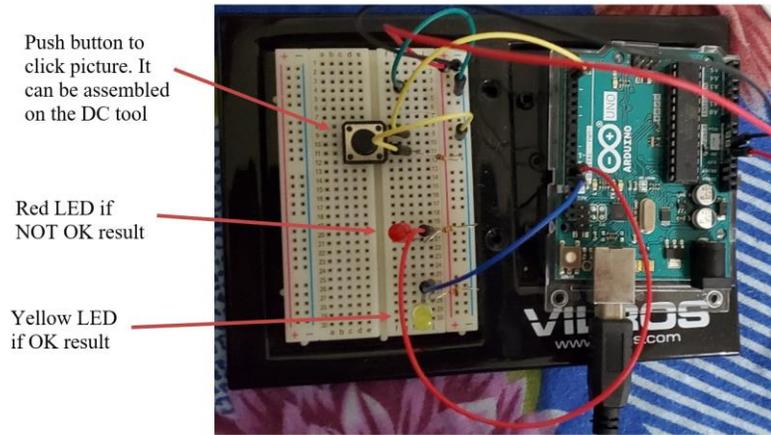


Figure 41. Arduino with push button to click picture.

The captured images for positions 1, 2, 3 and 4 are shown in Fig. 42 and the output for position 1 in the Python console is demonstrated in Fig. 43. A screenshot of the results text file is shown in Fig. 44.



Position 1



Position 2

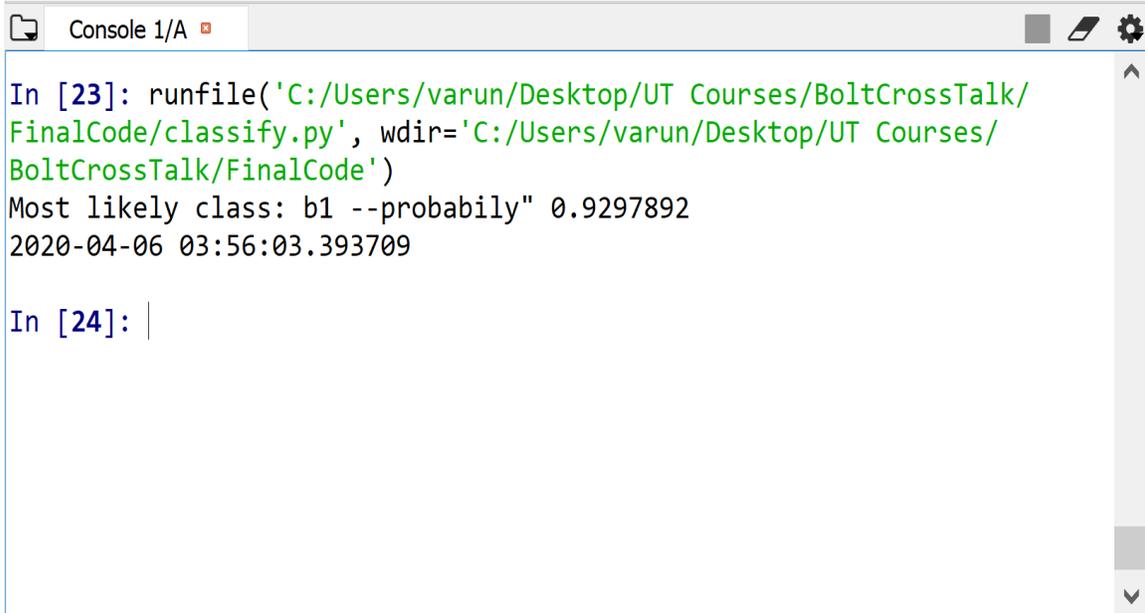


Position 4



Position 3

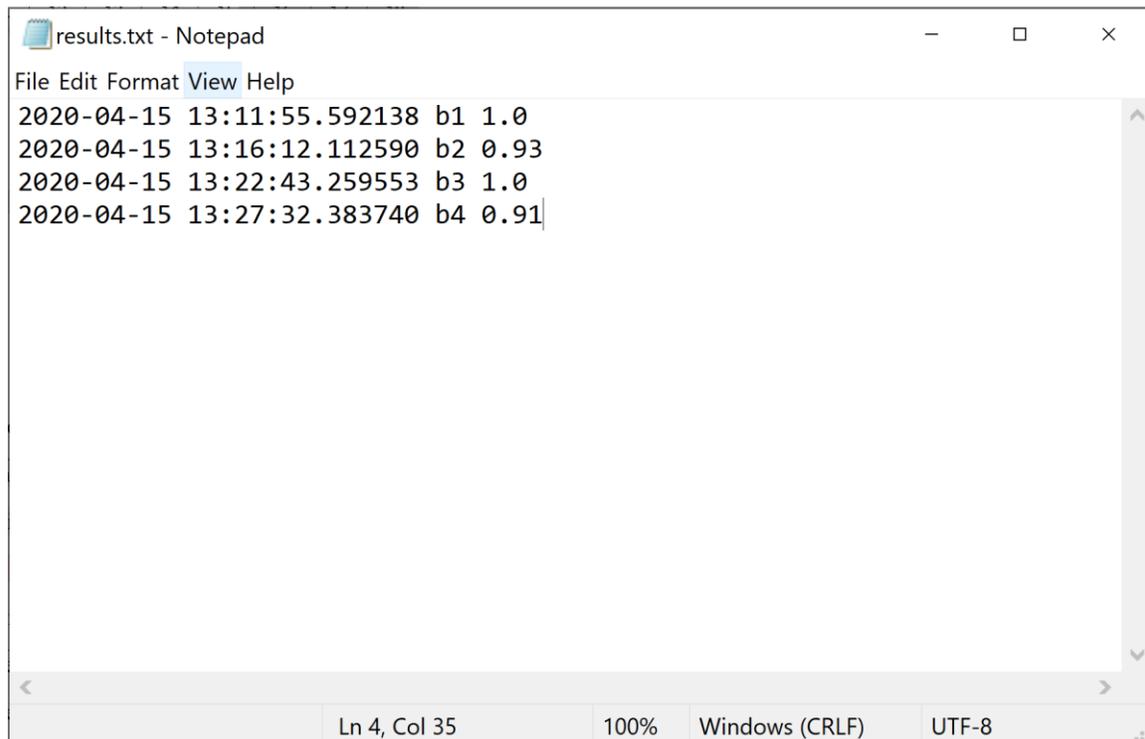
Figure 42. Captured images of all positions.



```
In [23]: runfile('C:/Users/varun/Desktop/UT Courses/BoltCrossTalk/
FinalCode/classify.py', wdir='C:/Users/varun/Desktop/UT Courses/
BoltCrossTalk/FinalCode')
Most likely class: b1 --probabily" 0.9297892
2020-04-06 03:56:03.393709

In [24]: |
```

Figure 43. Output in Python Console.



```
results.txt - Notepad
File Edit Format View Help
2020-04-15 13:11:55.592138 b1 1.0
2020-04-15 13:16:12.112590 b2 0.93
2020-04-15 13:22:43.259553 b3 1.0
2020-04-15 13:27:32.383740 b4 0.91|
```

Figure 44. Screenshot of output text file.

It can be observed in the output text file, the probabilities of predicting positions 1, 2, 3 and 4 are 1, 0.93, 1 and 0.91, respectively. As the trained model was able to predict all the positions correctly, the next step will be to mount the camera on DC tool and test the performance of system in an industrial environment. If the developed software is integrated with the DC tool or the torque wrench, the sequence for a particular model will be retrieved from the company's database hosted on the server via ethernet, when the product enters the station and actuates the limit switch. When the operator takes the tool to a position, if that position does not match the data received from the server, then the tool will not turn ON and the operator will not be able to tighten the bolt. Thus, the system will force the operator to follow the proper tightening sequence.

4.2 System testing in different lighting conditions

The effect of lighting conditions on the probability of detecting a position correctly was studied. As it is difficult to quantify the lighting conditions, the Python imaging library (PIL) module was used to change the brightness of the image to simulate different lighting conditions. The images shown in Fig. 42 were used as references and a factor of brightness was selected such that each image's brightness is varied from -100% to 100% in steps of 25%. The Python code to achieve this is as follows:

```
from PIL import Image, ImageEnhance
img=Image.open("temp\\tempImg.jpg")
img_brightness_obj=ImageEnhance.Brightness(img)
factor=int(input())
enhanced_img=img_brightness_obj.enhance(factor)
enhanced_img.save("temp\\tempImg.jpg")
```

Figure 45. Python code to change brightness of image.

4.2.1 Case 1: Brightness = -100%

The probabilities for positions 1, 2, 3 and 4 were found to be 1, 0.72, 1 and 0.87 respectively.

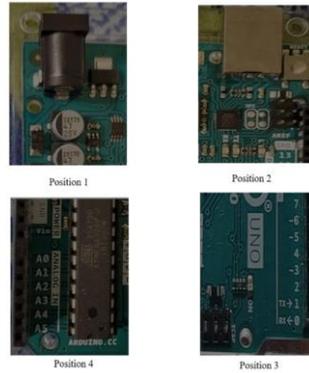


Figure 46. Case 1: Brightness= -100%.

4.2.2 Case 2: Brightness = -75%

The probabilities for positions 1, 2, 3 and 4 were found to be 1, 0.79, 1 and 0.88 respectively.

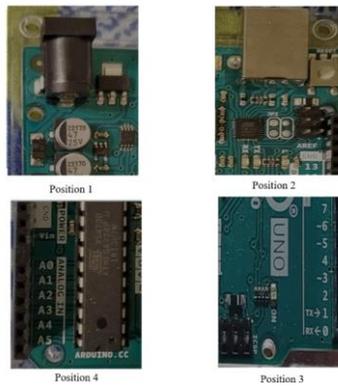


Figure 47. Case 2: Brightness= -75%.

4.2.3 Case 3: Brightness = -50%

The probabilities for positions 1, 2, 3 and 4 were found to be 1, 0.85, 1 and 0.88 respectively.

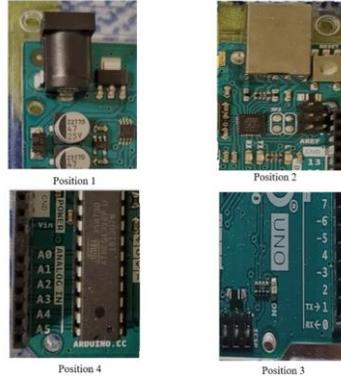


Figure 48. Case 3: Brightness= -50%.

4.2.4 Case 4: Brightness = -25%

The probabilities for positions 1, 2, 3 and 4 were found to be 1, 0.90, 1 and 0.89 respectively.

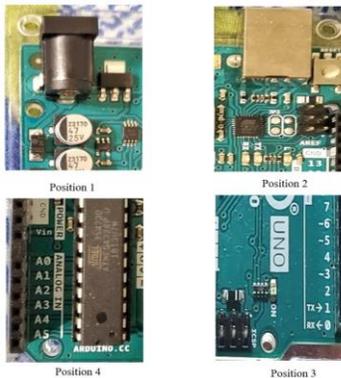


Figure 49. Case 4: Brightness= -25%.

4.2.5 Case 5: Brightness = 25%

The probabilities for positions 1, 2, 3 and 4 were found to be 0.99, 0.95, 1 and 0.87 respectively.

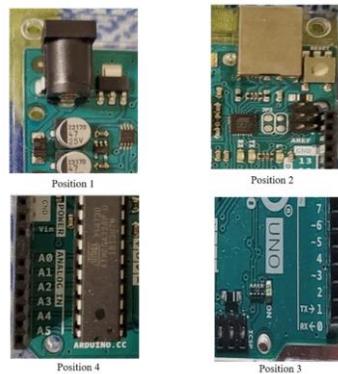


Figure 50. Case 5: Brightness= 25%.

4.2.6 Case 6: Brightness = 50%

The probabilities for positions 1, 2, 3 and 4 were found to be 0.99, 0.96, 1 and 0.87 respectively.

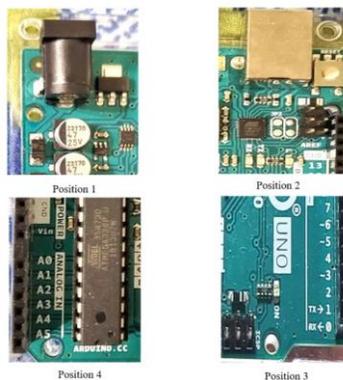


Figure 51. Case 6: Brightness= 50%.

4.2.7 Case 7: Brightness = 75%

The probabilities for positions 1, 2, 3 and 4 were found to be 0.98, 0.97, 1 and 0.86 respectively.

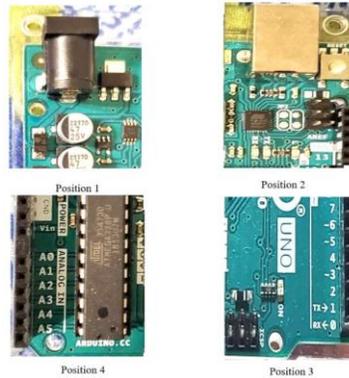


Figure 52. Case 7: Brightness= 75%.

4.2.8 Case 8: Brightness = 100%

The probabilities for positions 1, 2, 3 and 4 were found to be 0.96, 0.98, 1 and 0.86 respectively.



Figure 53. Case 8: Brightness= 100%.

4.3 Summary of results

The results from the nine cases reported above are summarized in Table 2. It was observed that the variation of probability of detecting a position with change in brightness does not follow the same trend for all the positions. For position 1, the probability decreased with increase in brightness and remained the same with decrease in brightness. For positions 2, the probability increased with increase in brightness. For position 3, the probability did not vary with brightness. For position 4, the probability decreased with increase in brightness as well as with decrease in brightness. (refer to Table 2 and Figure 54). Thus, it can be assumed that, depending on the characteristics of the features associated with each position, the brightness might affect the probability differently. But for all the lighting conditions, the software was able to detect each position correctly. The averages of the probabilities of detecting positions 1, 2, 3 and 4 were found to be 0.99, 0.90, 1.00 and 0.88 respectively. It should be noted that the algorithm was testing on a PC with lower specifications (see Appendix A), therefore time taken to detect each position was 30 seconds, which is too high to be implemented in manufacturing environment.

Brightness (%)	Probability of detecting a position			
	Position 1	Position 2	Position 3	Position 4
-100	1.00	0.72	1.00	0.87
-75	1.00	0.79	1.00	0.88
-50	1.00	0.85	1.00	0.88
-25	1.00	0.90	1.00	0.89
0 (Base)	1.00	0.93	1.00	0.91
25	0.99	0.95	1.00	0.88
50	0.99	0.96	1.00	0.87
75	0.98	0.97	1.00	0.86
100	0.96	0.98	1.00	0.86
Mean	0.99	0.90	1.00	0.88
Standard Deviation	0.01	0.09	0.00	0.02

Table 2: Summary of test results.

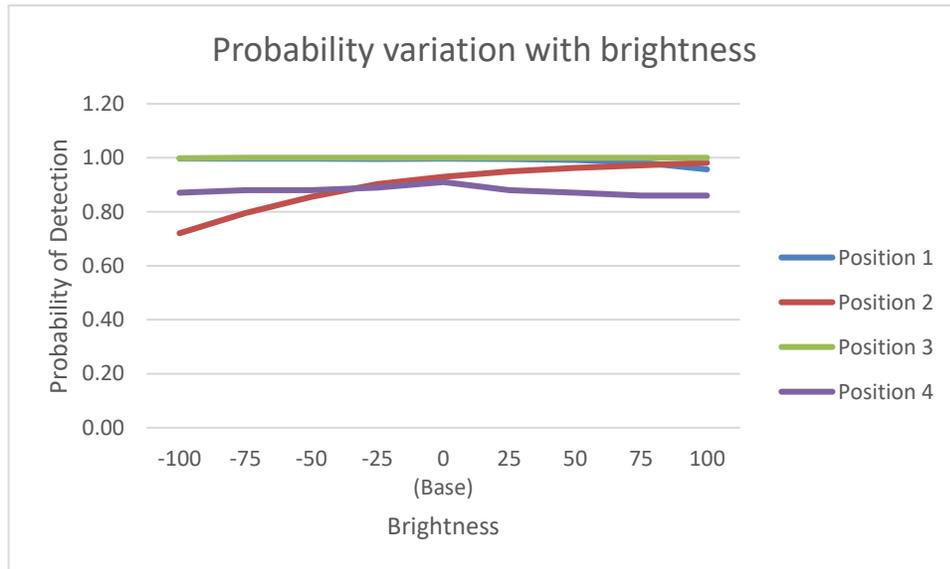


Figure 54. Chart showing summary of test results.

Chapter 5: Conclusion and Recommendations

5.1 Conclusion

The goal of this research was to develop a software tool that can be integrated with the DC tools and the torque wrenches used in manufacturing shops to make sure that the operator follows the bolt tightening sequence defined by the engineering specifications. In addition, the system should be able to help map and record torque values with each bolt, where there are multiple similar bolts that need to be tightened up to different torque values on the same station. The software developed in this work can identify each bolt position using a unique characteristic related to it. As the developed software has not been interfaced with DC tool and the company's database, it should be noted that the software developed in this work does not map torque values with each bolt. Unlike National Instruments' AI Vision Builder, for which the operator needs to have technical expertise of the software to set up the program and to test different algorithms [30], the developed program is comparatively easy to set up and the operator does not need to test different algorithms for an application. . The software developed in this research can find its application in all the industries where torqueing sequence is essential for quality.

For the experimental conditions tested, the average of probabilities of detecting a position under different experimental conditions was observed to be more than 85%. The use of the Keras library to apply image transformation reduces the size of the dataset needed, thus resulting in a decrease in the time for data collection. The software developed in this research was able to detect all the test cases correctly, thus achieving the accuracy of 100%.

5.2 Improvements and Recommendation

The research was more focused on bolt location recognition and optimizing the algorithm parameters, therefore integration with the DC tool was not studied. The next step would be to mount camera on a DC tool and integrate the software with DC tool's controller to test performance by tightening automotive assemblies in realistic conditions and developing the algorithm to select different communication protocols based on DC tool manufacturer.

Integration with the database of a company was not done because each company has specific standards of data communication and employ different technologies. When this software is integrated with DC tool, this system can be deployed on the company's manufacturing execution system (MES) to study the performance of the system in manufacturing environment.

The time taken to detect one location is around 30 seconds because of hardware limitations of the PC used in this research. Specifications of the PC used in the research can be seen in Appendix A. Thirty seconds is too long for use in practice. Based on the author's experience with various automotive industries, the average cycle time in a car's assembly shop is 52 seconds and each station has at least 4 different bolt locations, therefore the system should be able to detect each location in less than 1 second. The execution file of program was run on a PC with 16 GB of RAM and 8 GB graphics card (Appendix C) and the time for detecting one position was observed to be 0.53 seconds.

Appendices

Appendix A: PC SPECIFICATIONS [27]

OPERATING SYSTEM	Windows 10 Home 64 [20]
PROCESSOR	Intel® Core™ i7-8565U (1.8 GHz base frequency, up to 4.6 GHz with Intel® Turbo Boost Technology, 8 MB cache, 4 cores) [16,17]
GRAPHICS	Integrated: Intel® UHD Graphics 620
DISPLAY	13.3" diagonal FHD IPS micro-edge WLED-backlit touch screen with Corning® Gorilla® Glass NBT™ (1920 x 1080) [12]
DISPLAY BRIGHTNESS	400 nits
DISPLAY COLOR GAMUT	72% NTSC
MEMORY	8 GB DDR4-2400 SDRAM (onboard)

Appendix B: ARDUINO UNO SPECIFICATIONS [28]

- Microcontroller: ATmega328P
- Operating Voltage: 5V
- Input Voltage (recommended): 7-12V
- Inout Voltage (limit): 6-20V
- Digital I/O Pins: 14 (of which 6 provide PWM output)
- PWM Digital I/O Pins: 6
- Analog Input Pins: 6
- DC Current per I/O Pin: 20 mA
- DC current for 3.3V Pin: 50 mA
- Flash Memory: 32 KB (ATmega328P) of which 0.5 KB used by bootloader
- SRAM: 2 KB (ATmega328P)
- EEPROM: 1 KB (ATmega328P)
- Clock Speed: 16 MHz
- LED_BUILTIN: 13
- Length: 68.6 mm
- Width: 58.4 mm
- Weight: 25 g

Appendix C: RECOMMENDED PC SPECIFICATIONS [29]

Processor	7th Gen Intel Core i7-7820HK (8MB Cache, 2.9GHz - 3.9GHz)
Display	17.3-inch WQHD (2560x1440) 120 Hz, TN Anti-Glare, G-SYNC, LED-Backlit Display
Graphics	Nvidia GeForce GTX 1080 (8GB GDDR5X)
Memory	16GB 2400MHz DDR4 Memory
Storage	512GB NVME SSD + 1TB HDD
Optical	NA
Wired Ethernet	Killer Gigabit Ethernet
Wireless Connectivity	Killer 802.11ac Dual-Band 2.4GHz & 5GHz, BT 4.1
Interface (Left)	Lock Slot, USB-C Port, USB 3.0 Port, Mic Jack, Headphone Jack
Interface (Right)	USB 3.0 Port
Interface (Back)	Ethernet Port, Mini-DisplayPort, HDMI Port, Thunderbolt 3.0 Port, Graphics Amplifier Port, Power Connector
Operating System	Windows 10 Home 64-bit
Dimensions	16.7 x 13.1 x 1.18 inches (W x D x H)
Weight	9.74 pounds
Extras	Tobii IR Eye-Tracking presence detection, 99Whr Battery
Warranty	1-Year Mail-In Service

References

- [1] “Tightening Sequences”, Tightening Sequence for a Joint Consisting of Several Bolts. Last accessed 30th Aug 2019: www.boltscience.com/pages/tsequence.htm.
- [2] “Specify a Torque & Tightening Sequence for Critical Fastening Joints”, Mountz Torque, 30 Nov. 2018. Last accessed 30th Aug 2019: www.mountztorque.com/Specify-a-Torque-Tightening-Sequence-for-Critical-Fastening-Joints.
- [3] “Southwest Engines How Engines Work: Engineering, Car Engine, Automotive Mechanic.” Last accessed on 30th August, 2019: www.pinterest.ca/pin/310255861811747387/.
- [4] Vukelic, D., Ostojic, G., Stankovski, S., Lazarevic, M., Tadic, B., Hodolic, L. and Simeunovic, N. (2011), “Machining fixture assembly/disassembly in RFID environment”, *Assembly Automation*, Vol. 31, No. 1, pp. 62-68.
- [5] Rusli, L. and Luscher, A. (2012), “Fastener identification and assembly verification via IR tracking”, *Assembly Automation*, Vol. 32, No. 3, pp. 262-275.
- [6] Killing, J., Surgenor, B.W. and Mechefske, C.K. (2009), “A machine vision system for the detection of missing fasteners on steel stampings”, *International Journal of Advanced Manufacturing Technology*, Vol. 41, No. 808.
- [7] Rusli, Leonard, and Anthony Luscher, “Fastener Identification and Assembly Verification via Machine Vision”, *Assembly Automation*, Vol. 38, No. 1, 2018, pp. 1–9.
- [8] Karpathy, Andrej. 2015. “Neural Networks Part 1: Setting Up the Architecture.” Notes for CS231n Convolutional Neural Networks for Visual Recognition, Stanford University. Last accessed on 30th January, 2020: <http://cs231n.github.io/neural-networks-1/>.
- [9] “Convolutional neural network.” Wikipedia. Last accessed on 5th Feb, 2020: https://en.wikipedia.org/wiki/Convolutional_neural_network
- [10] Sermanet, Pierre, and Yann LeCun. 2011. “Traffic Sign Recognition with Multi Scale Networks.” Courant Institute of Mathematical Sciences, New York University. Last accessed on 8th Feb, 2020: <http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=6033589>.

- [11] Ovtcharov, Kalin, Olatunji Ruwarse, Joo-Young Kim et al. Feb 22, 2015. “Accelerating Deep Convolutional Networks Using Specialized Hardware.” Microsoft Research. <http://research.srv.microsoft.com/pubs/240715/CNN%20Whitepaper.pdf>
- [12] “Convolutional neural network,” Wikipedia. Last accessed on 3rd April, 2020: https://en.wikipedia.org/wiki/Convolutional_neural_network
- [13] Samer Hijazi, Rishi Kumar, and Chris Rowen. (2015), “Using Convolutional Neural Networks for Image Recognition [PDF file]”. Cadence Systems. Last accessed on 4th April, 2020: https://ip.cadence.com/uploads/901/cnn_wp-pdf
- [14] De Ruvo, P. (2009), “A GPU-based vision system for real time detection of fastening elements in railway inspection”, 16th IEEE International Conference on Image Processing (ICIP), Cairo, pp. 2333-2336.
- [15] Emhart Americas Inc. (2011), “Stanley assembly technologies DC electric tools”. Last accessed on 5th April, 2020: www.emhartamericas.com/brands/stanley-assembly-technologies/products/threadedfastening/electric (accessed 21 December 2011).
- [16] Feng, H., Jiang, Z., Xie, F., Yang, P., Shi, J. and Chen, L. (2013), “Automatic fastener classification and defect detection in visionbased railway inspection systems”, IEEE Transactions on Instrumentation and Measurement, Vol. 63, No. 4, pp. 877-888.
- [17] Gonzalez, D., Botella, G., Meyer-Baese, U., Garcia, C., Sanz, C., Prieto-Matias, M. and Tirado, F. (2012), “A low cost matching motion estimation sensor based on the NIOS II microprocessor”, Sensors, Vol. 12, No. 10, pp. 13126-13149.
- [18] Hage, P. and Jones, B. (1995), “Machine vision-based quality control systems for the automotive industry”, Assembly Automation, Vol. 15, No. 4, pp. 32-34.
- [19] Igual, F.D., Botella, G., Garcia, C., Prieto, M. and Tirado, F. (2013), “Robust motion estimation on a low-power multicore DSP”, EURASIP Journal on Advances in Signal Processing, Vol. 2013, pp. 99.
- [20] Killing, J., Surgenor, B.W. and Mechefske, C.K. (2009), “A machine vision system for the detection of missing fasteners on steel stampings”, International Journal of Advanced Manufacturing Technology, Vol. 41, No. 808.
- [21] Perkins, W.A. (2009), “INSPECTOR: a computer vision system that learns to inspect parts”, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 5, No. 6, pp. 584-592.

- [22] Poljak, J., Botella, G., Garcia, C., Poljacek, S.M., Prieto-Matias, M. and Tirado, F. (2013), "Offset printing plate quality sensor on a low-cost processor", *Sensors*, Vol. 13, No. 11, pp. 14277-14300.
- [23] Simpson, L. (2003), "Machine vision improves productivity in many ways", *Assembly Automation*, Vol. 23, No. 3, pp. 243-248.
- [24] Wang, F., Shen, B., Sun, S. and Wang, Z. (2016), "Improved GA and Pareto optimization-based facial expression recognition", *Assembly Automation*, Vol. 36, No. 2, pp. 192-199.
- [25] Welch, G. and Foxlin, E. (2002), "Motion tracking: no silver bullet, but a respectable arsenal", *IEEE Computer Graphics and Applications*, Vol. 22, No. 6, pp. 24-38.
- [26] "Select Your Vision Builder for Automated Inspection License", National Instruments. Last accessed on 15th April 2020: www.ni.com/en-us/shop/electronic-test-instrumentation/application-software-for-electronic-test-and-instrumentation-category/what-is-vision-builder-for-automated-inspection/select-license.html.
- [27] "HP Spectre x360 - 13-ap0039nr", Hewlett-Packard Company. Last accessed on 4th May 2020: store.hp.com/us/en/pdp/hp-spectre-x360-13-ap0039nr.
- [28] "Arduino Uno Specification," TOMSON ELECTRONICS. Last accessed on 4th May 2020: www.tomsonelectronics.com/blogs/news/arduino-uno-specification.
- [29] "Alienware 17 R5 Review", TechRadar. Last accessed on 4th May 2020: www.techradar.com/reviews/alienware-17-r5.
- [30] "NI Vision Builder for Automated Inspection Tutorial", National Instruments. Last accessed on 4th May 2020: <http://www.ni.com/pdf/manuals/373379h.pdf>.