Automated Scratch Detection System For The Automotive Industry

The Capstone Design Project
Presented to the academic Faculty

By

Syed Muhammad Hasan Naqvi

Saira Khan

Muhammad Ashir Wahid

In partial fulfillment of the requirements for
B.S. Electrical Engineering
In the
School of Science and Engineering

Habib University

April 2018

Automated Scratch Detection System For The Automotive Industry

The capstone design project was advised by:

_____

Dr. Muhammad Farhan
Faculty of Electrical Engineering
Habib University

Approved by the Faculty of Electrical Engineering on April 13, 2018

We cannot solve our problems with the same thinking we used when we created them.

Albert Einstein

For Habib University.

Acknowledgements

First and foremost, we would like to express our appreciation to the institutions and the individuals that made possible the very idea of Habib University. We are grateful to our community of donors for making it possible for us to have a pedagogical experience like no other.

A very special acknowledgement for our project supervisor, Dr.Muhammad Farhan, who has been a phenomenal mentor and supporter throughout this tough journey. Thanks to his presence and support, we have been able to grow and learn beyond our limits. His expertise and excellent mentorship has been contributing to our core motivation to pursue this project ever since the first day.

Moreover, we would like express our gratitude to Dr.Basit Memon and Mr.Tariq Mumtaz who have been present to advise us on everyday matters whenever necessary. Their constant understanding and support has gotten us a long way.

We are extremely grateful for the presence of Masab Iqbal, Zeeshan Nafeez, and Irfan Muhammad, who have been guiding and supporting us through their interest in our work and growth.

Furthermore, a huge shout out to Talal Wasim for helping us with our work through his own learnings and knowledge.

Finally, we are thankful to our friends for being a support system that we could rely on with everyday downfalls and difficulties. Thank you to Sana Rizwan Gondal, Ambreen Aslam, Osama Bin Rizwan, Hasnain Raza, Tabish Azam, Shafaat Khuwaja, and Anushay Zehra Rashid.

Executive Summary

Following the manufacturing process within an automotive industry, a vehicle is prone to defects being present on its body. In the Pakistani industry, one of the largest contributors to these defects are scratches. The current measure being taken to detect scratches on vehicles involve the presence of trained specialists who inspect the body under specific lighting conditions.

For this very reason, a low cost and effective automated system is desired which is capable of detection scratches on the front bumper of vehicles on the production line of industries. The main features of this system should its ability to pinpoint a range of areas where the scratches originate from in the manufacturing process as well as to help improve the current manual inspection system in place.

To tackle the problem at hand, we propose a scratch detection device, based on a microcontroller interfaced with a distance sensor and a camera module, set up on a robotic arm which is mapped to the front bumper of a vehicle. The system is placed on the manufacturing line. It detects the presence of a vehicle through the threshold being broken on the distance sensor. It then begins the mapping process and takes images of the bumper which it then sends to the image processor located on the microcontroller of the device.

The image processor loads the images and begins to preprocess them. This involves the removal of unwanted objects, the removal of reflections of surfaces, etc. It then passes the preprocessed image onto a trained machine learning model which outputs a decision stating whether a scratch is present or not. This data is then logged onto a spreadsheet online which is accessible by the human inspector.

The trained machine learning model is based on a Convolutional Neural Network that is trained on a image dataset of around 1500 images of scratches and non-scratches. This model has an accuracy rate of 90% and is capable of telling whether a scratch is present in an image or not.

The Robotic Arm has 6 degrees of freedom, allowing it to move in all four directions in a spacial plane as well as extend further into space. This is done to take into account the curved body of a front bumper.

The proposed implementation is to place two such Automated Scratch Detection Systems on two sides of the vehicle at a distance of 16 centimeters from the body. This ensures optimal processing time as well as detection accuracy. Each of the systems map towards one half of the bumper and perform the image processing task in a total of 12 minutes.

Table of Contents

Introduction

The automotive industry in Pakistan is one of the largest growing industries that produces over 180,000 vehicles in a given year [1]. Each of these vehicles go through a manufacturing process, and in order to ensure high customer satisfaction, the minimization of defects on the vehicle following the production process is necessary.

The manufacturing process begins at the Press Shop, where different parts of the body are created and pressed. These parts are then put through to the Weld Shop where the core body of the car is put together. The body then goes into the Paint Shop where the color of the vehicle is applied. Finally, the body arrives at the Assembly Shop where the remaining parts are added to it.

The Assembly Shop is divided into three main lines: Trim Line, Chassis Line, and the Final Line. At the Trim Line, the interior components of the car are installed. These range from wiring, locking mechanism, dashboard, and so on. At the Chassis Line, the car is lifted upwards and the engine, suspension, transmission, base frame, and other such parts are installed. Finally, at the Final Line the remaining features are put together such as the tires, windshield, side mirrors, etc.

Once this process is complete, the vehicle is sent for testing where the inspection of defects happen. As of now, the current process in place in most industries in Pakistan is a manual human eye inspection, where trained specialists observe the vehicle body under high intensity lighting to note down the defect's data on a sheet of paper.

Defects Per Unit (DPU) - the average number of defects in a given sample of vehicles. The types of defects are: missing component, fitting issues, paint anomalies, scratches, and dents. The last two types, namely scratches and dents, contribute to around 60% of the overall defects being brought about on cars at a certain vehicle production industry in Pakistan.

The larger contributor to defects are scratches on a vehicle which come about at some point during the manufacturing process on the Assembly Line. The current system to detect scratches is efficient with the successful detection of all scratches, however, it is extremely time consuming and is unable to tell at which point the scratches come about on the vehicles.

Problem Statement

The problem at hand is that the automotive industry is looking for an automated solution to be able to detect scratches on their vehicles. The main aim is to be able to narrow down the responsibility to pinpoint which line in the Assembly Shop is contributing to the scratches on vehicles. In addition to that, the idea is to be able to implement a solution that helps the current system in place detect scratches in a quicker time period.

The problem statement can be stated as follows:

"A low cost and effective system for detecting scratches on the front bumper of vehicles at the Assembly Line (Production Department) of Indus Motors Company"

The key factors or constraints that we need to take into account with the system are:

- *Narrow down the responsibility*: The system needs to be able to pinpoint exactly at which part of the Assembly Line (Trim Line, Chassis Line, or Final Line) did the scratch come about so the particular part of the production process can be held responsible
- *Help improve current inspection system*: We also need to ensure that the current system, which involves human inspectors observing scratches and manually noting them down, is improved and can use assistance through our potentially solution.

While the problem at hand is extremely broad, we have narrowed down our problem of interest and are choosing to work with developing an automated solution towards effective scratch detection on the front bumper of vehicles in the automotive industry.

Scratch Characteristics

The scratches that we are dealing with are not just any other scratches. These are minute and small anomalies that come up on vehicles during the manufacturing process. The characteristics of the scratches, given by the automotive industry, are stated as follows:

- A defect is counted as a scratch if it is at least 5 millimeters in length
- They are barely visible by the naked eye
- They are observed by trained experts using specific lighting conditions

Design Details

Benchmarking

Different automobile industries from all over the world have moved towards automation when it comes to areas like scratch detection. In our research, we were able to find out that different systems are in place of which there are no details available except for a general overview.

Industries in Japan have used cameras set up at their production lines to take images of cars and apply image processing techniques to detect any anomalies on the body. Similarly, in Thailand there are video recording systems in place which perform similar tasks.

Possible Solutions

When researching possible solutions, we came across various options that could be considered feasible. One of these solutions was the use of a system which provides fully automatic

detection and recognition of minor vehicle body damages, which utilizes a sensor network integrated into the vehicle body. The efficiency of this system depends on the acoustic vehicle noise. The implementation of the system uses Piezoelectric Sensor Elements.

Piezoelectric Sensor Elements are based on polyvinylidene fluoride foils. These are bonded on the housing of the sensor nodes within the internal structure of the car body. The sensors measure vibrations and damages in terms of their structure-borne sound. The sensor elements generate a voltage proportional to the mechanical vibrations in the vehicle body. The analog signals are filtered by a low pass filter, and digitized by an analog to digital converter. The digitized time signals are then transferred to a microprocessor-based embedded system, which processes the damage data based on the implemented algorithms and the information a parent instance provides.

Within the sensor node network of a vehicle, eleven Piezoelectric Sensor Nodes are used.

Figure 1 – Sensor Node Positions

The other possible solution involved the use of cameras installed at different locations on the assembly line. These cameras would take images of the vehicles and process them for scratches using image processing techniques. Such a placement map will allow us to narrow down the defect responsibility. We will be taking images of the car before it enters the Trim Line and after it leaves the Trim Line. Similarly, this will be done at the Final Line too. With this methodology, we would be able to apply inspection through image processing techniques to check whether the responsibility of the scratch falls on the Trim Line, Chassis Line, Final Line, or some point before the Assembly Line.

However, while this solution seemed feasible as compared to installing sensors on the vehicles. We still chose to refine it further in order to propose a more efficient solution to the problem at hand. Our proposed solution is discussed in the next section.

Proposed Solution

Following our market analysis and our in-depth survey of the assembly line at the industry, we propose a scratch detection device which is based on a microcontroller interfaced with a camera module and a distance sensor. This device will hold an image processing software based on a Machine Learning model which will be able to classify images into scratches and non-scratches. The device will be mounted on a robotic arm whose positions will be mapped to the front bumper of a vehicle. The system, placed on the Assembly Line, will be able to detect the presence of a vehicle through the distance sensor, following which the robotic arm will move the device throughout the bumper as images are taken and processed. The processed data will then be uploaded to a spreadsheet online to be viewed by the industry personnel.

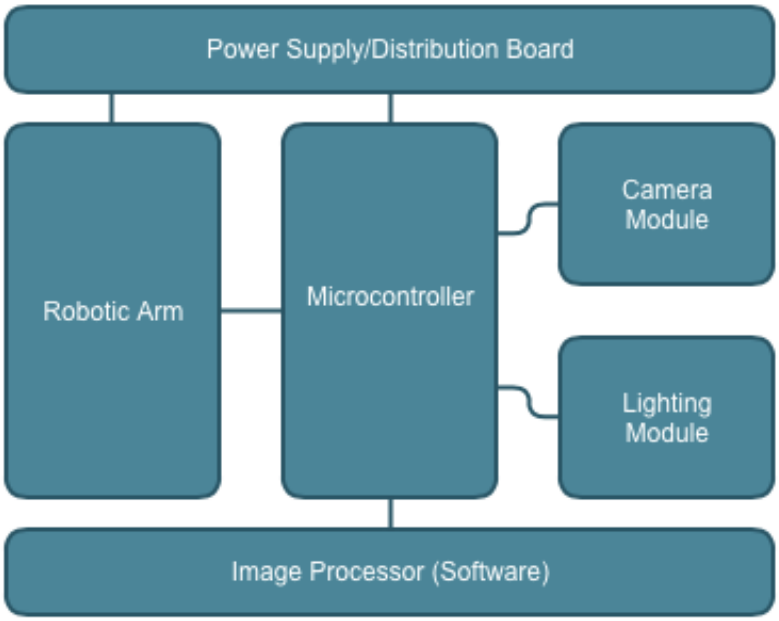


Figure 2 – Block Diagram Of Proposed Solution

The key technical specifications of our solution are as follows:

- Our solution will be able to operate on both automatic and manual modes. When set on automatic, the device will be interfaced with the robotic arm where it will be able to take images of the bumper as per the arm mapping once the distance sensor detects the presence of a vehicle. These images will go through the image processor following which the scratch data will be uploaded onto a spreadsheet. On the other hand, while operating on manual mode, the device becomes handheld and can be used to manually detect a scratch on a certain part with the click of a button. A similar procedure is followed following the image taking process of a single capture.

- Our solution will be able to detect scratches based on a 85% accuracy. The idea is to be able to develop a system that will assist the current system to perform better. Our

machine learning model will be based on Convolutional Neural Networks which will be able to tell the presence of a scratch or not, with an accuracy of around 85%.

- Our solution will be able to log the scratch data onto a spreadsheet to ensure that the data is readily available to be used by the personnel. Moreover, on a given vehicle, it will be able to tell the number of scratches that are present and will assist the trained experts in analyzing the vehicle better.

Our proposed solution can be broken down into the following subsystems, as shown in Figure 3.

Figure 3 – Proposed Solutions Subsystems

Subsystem I – Robotic Arm

Motivation & Research

An industrial robot arm is a marvel of engineering in that it reacts similarly to our own arms. The arm resembles exactly like a human arm, with a forearm, elbow and shoulder. The six axis robot has six degree of freedom, allowing it to move in six different ways, compare it with a human arm which has seven degree of freedom. The purpose of robotic arm in our project is to automate our system, remove human intervention and speed the working process.

Keeping the design of the front bumper in mind, we needed to come up with a robotic arm that could move in all four directions in a two-dimensional plane, as well as extend further into the space to reach out to the front of the vehicle if the system is placed on the side. The vehicle that needs to be inspected have a curved surface, due to this reason we designed a Robotic Arm with six Degree of Freedom (6DoF). Extension of the ARM are designed as the prototype which can cover extend half of the front bumper. In this regard a 3D model of Robotic was designed with a six degree of freedom (6DoF). Figure 4 shows the robotic arm model which we have adopted for this project, inspired by [1].

The Robotic Arm implementation can be divided into two parts, namely the turntable and the arm itself. The turntable provides the system with a complete rotational movement ranging

from 0 to 180 degrees. On the other hand, the arm provides the system with extensions and adjustability between the distance of the vehicle and other spacial dimensions.

Figure 4 – Robotic Arm 3D model

Specifications Of The Robotic Arm

The specifications of the Robotic Arm subsystem are stated as follows:

- The Robotic Arm system has 6 degrees of freedom
- It's complete length, during maximal extension, is 45 centimeters
- The body of the Robotic Arm is printed using a 3D printer with Polyactic Acid
- The system is powered using 6V and 6.5A supply for the motors
- The system utilizes five MG995 Servo Motors and one 6 pole Stepper Motor
- The overall system is interfaced with the Arduino Mega 2560 for operation

Components Required For The Robotic Arm

Item	Quantity
6-pole Stepper Motor	1
10mm Steel Balls	10
Motor Driver L298N	1
Standard Size TowerPro MG995 Servo Motor	5
Veroboard (5cm x 10cm)	1

Power Supply (6V & 6.5A)	1
Arduino Mega 2560	1
Ultrasonic Distance Sensor HC-SR04	1

Stepper Motor – Working Principle

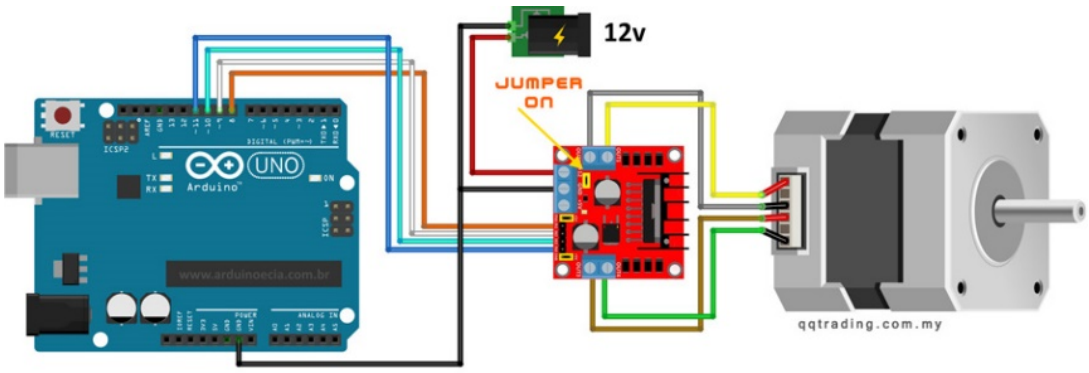


Figure 5 – Stepper Motor Connections With Arduino

Stepper Motor is a brushless DC motor that rotates in steps. We choose a stepper motor because we can precisely give positions to the motor without any feedback sensor, which represents an open-loop controller. A 2 pole stepper motor consist of a permanent magnet and it is surrounded by the windings of the stator. To run a stepper in a specific position we need to activate the windings step by step in a particular order and let a current flow through them they will magnetize the stator and make electromagnetic poles respectively which causes a propulsion to the motor. We use a Full step drive mode using L298N motor driver, full step drive mode provides much higher torque output as it has 2 coils active at a same time.

Interfacing HC-SR04 With Arduino

Distance Sensor is implemented on the base of the arm. The purpose of distance sensor is to calculate the distance from the car body. Whenever the distance between the car and the arm will break a certain predefined threshold, Arduino will give commands to the motors to start mapping the area of the bumper. Below is the schematic showing the connection of distance sensor from the Arduino.

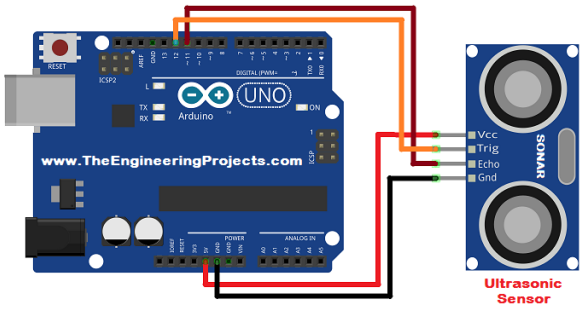


Figure 6 – Interfacing HC-SR04 with Arduino

Connecting Servo Motors With Arduino

A set of three extensions are made in order to get the desired strength in the Robotic Arm, these extensions are being supported by 5 servos which provide extension to the arm and rotation to the device which is installed at the top of the arm.

A set of 5 servo motors are used in order to provide arm the strength to distribute its weight accordingly. Each servo has the capability to rotate from a range of 0 to 180 degree. We split our servo into four sections namely base servo, lower servo, upper servo and device servo.

The base servo contains two servo's which is mounted onto the turntable, these two servo draw the most current as most of the weight of the arm is on these servo's. Lower servo is mounter at the end of the first arm which is extended by a single extender. Upper servo holds the smallest extender which are mainly used to make the device stable. Device servo is used to adjust the camera location facing the vehicle. Each servo is controlled by an Arduino Mega, and each section of positioning is mapped accordingly to get the maximum positioning in evaluating a Vehicle. Each position is followed by a rotation made by the stepper motor.

The interfacing of all the motors, motor driver, and sensor with the Arduino can be visualized through the circuit diagram as shown in Figure 7. The Robotic Arm is supplied power through a power supply system with all the grounds between the equipment and the microcontroller being shorted.

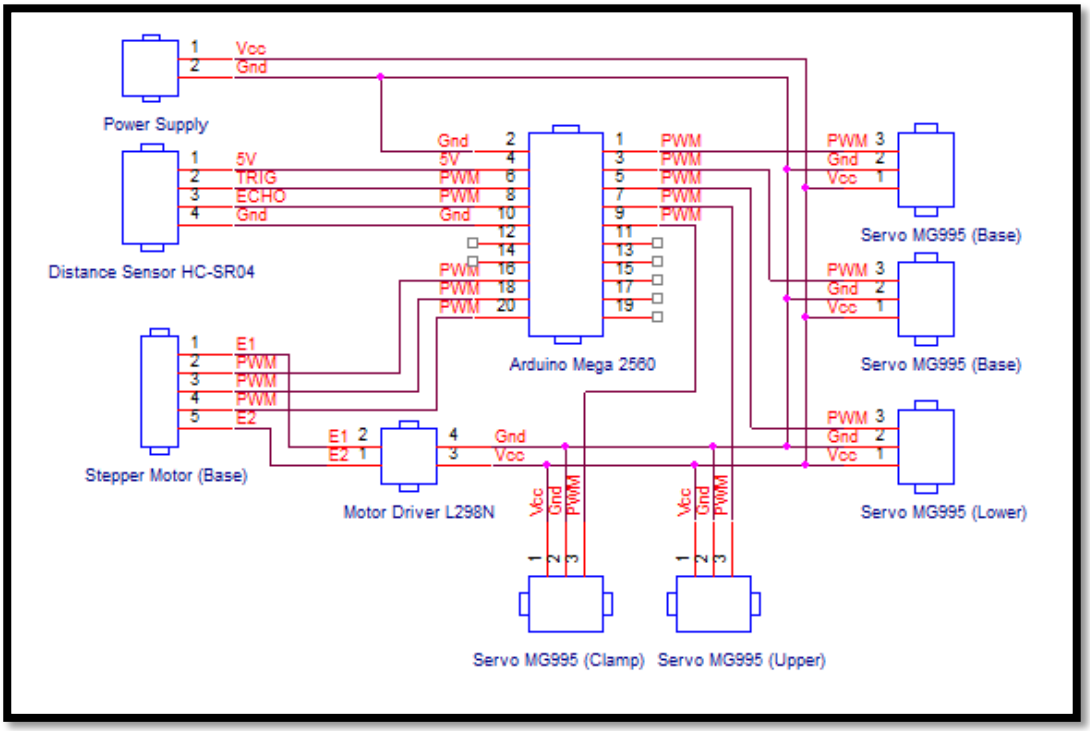


Figure 7 – Robotic Arm Circuit Diagram

Subsystem III – Scratch Detection Device

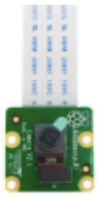
The Scratch Detection Device is the brains behind our automated solution. It holds the image processor software and contains the hardware used to capture the images of a vehicle's body. This subsystem is basically a hand-held device based on a microcontroller, that can operate at two different states, automatic or manual, and is interfaced with a camera and a distance sensor.

Choosing A Microcontroller – Raspberry Pi 3

The choice of hardware for this subsystem can be depicted in Figure 8.

Raspberry Pi 3
Quad Core 1.2GHz Broadcom CPU 64 bits
1 GB RAM

HC-SR04 Ultrasonic Distance Sensor



Raspberry Pi Camera Module V2
8 Megapixels
Sensor: Sony IMX219
Sensor Resolution: 3280 x 2464 pixels

Figure 8 – Subsystem I - Hardware

For the choice of our microcontroller, we needed a system which had enough processing power to be able to handle image processing and machine learning techniques. Given that our system would be taking images, processing them, and making decisions, we needed a microcontroller which could hold handle such tasks. For this very reason, we opted for the Raspberry Pi 3 as it is equipped with a Quad Core 1.2GHz Broadcom processing unit, 1GB RAM, and other memory options that allowed us to be able to process large image processing data easily.

Camera Selection

Once we had a microcontroller selected, we needed a camera which would be able to capture high quality images of the vehicle's body. A constraint that we had to keep in mind was that our camera should be compatible with our microcontroller. Our research presented us with a diversity of options, which is when we decided to perform an analysis on the specifications versus the cost to understand what the optimal solution would be.

We narrowed down our searches to the Raspberry Pi Camera Modules as we believed it would be best to interface two systems from the same producers for optimal functionality. For this

reason, we looked into the Raspberry Pi Camera Module V2 which had outstanding specifications given its cost. The detailed features of this module can be shown in Figure 9.

	Camera Module v1	Camera Module v2
Net price	$25	$25
Size	Around 25 × 24 × 9 mm	
Weight	3g	3g
Still resolution	5 Megapixels	8 Megapixels
Video modes	1080p30, 720p60 and 640 × 480p60/90	1080p30, 720p60 and 640 × 480p60/90
Linux integration	V4L2 driver available	V4L2 driver available
C programming API	OpenMAX IL and others available	OpenMAX IL and others available
Sensor	OmniVision OV5647	Sony IMX219
Sensor resolution	2592 × 1944 pixels	3280 × 2464 pixels
Sensor image area	3.76 × 2.74 mm	3.68 x 2.76 mm (4.6 mm diagonal)

Figure 9 – Raspberry Pi Camera Module V2 Specifications

Other Components

Finally, we required a distance sensor which would be able to tell us if a vehicle is present on a given location on the assembly line where our system will be in place. For this, our cheapest and most efficient solution was the Ultrasonic Distance Sensor HC-SR04. This was available in the local market for a cheap price and was easily compatible with our Raspberry Pi 3.

In addition to that, we bought other minor items such as a two-way switch, push button, and resistors. These were necessary in order to ensure our device begins to function as desired.

Interfacing The Hardware

After purchasing our hardware, we went ahead to interface all the equipment together. For this task, we designed a Printed Circuit Board (PCB) which can be seen in Figure 10. The PCB

included Header 4 representing the HC-SR04 sensor, Header 5 representing the Raspberry Pi 3 GPIO pins, Header 2 representing the push button switch, and finally Header 5x2 representing the two-way switch. The successful interfacing of the equipment with the PCB allowed the hardware of our system to be completed.

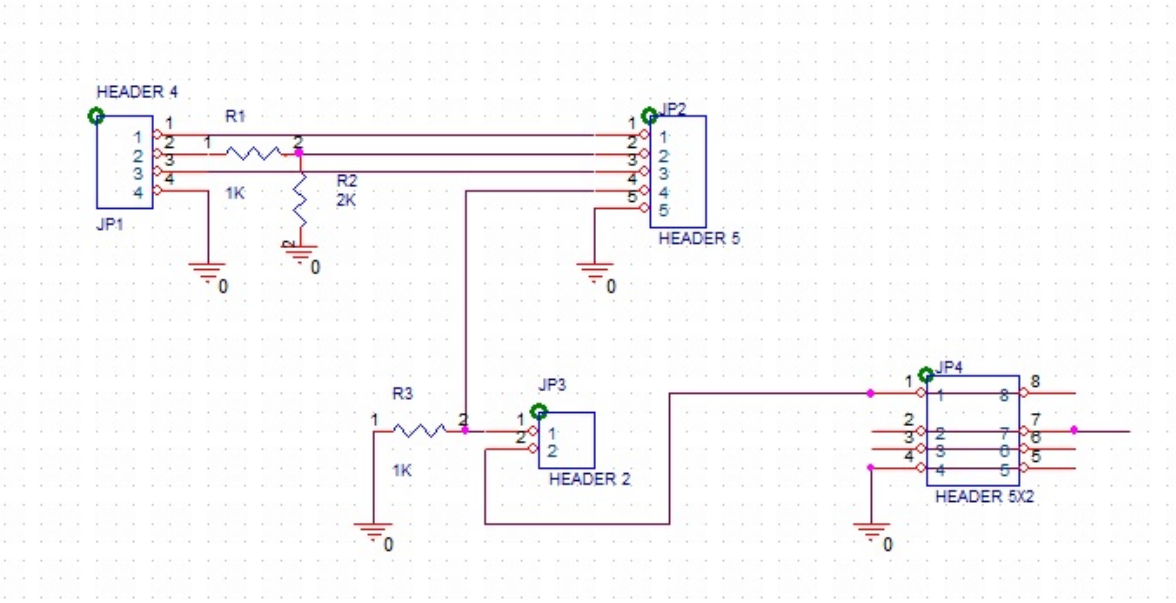


Figure 10 – Scratch Detection Device Circuit Diagram

Designing The User Interface

No system is complete without a software behind it. For this very reason, our Scratch Detection Device contained one of the most important parts of our project, the image processing system. The programming of the software was done on Python, and the different modules can be seen in Figure 11 that take a flow of direction from left to right.

Figure 11 – Subsystem I – Software

The User Interface module brings together the overall software system. It begins with detection the mode of the device, being either manual or automatic. If the mode is set to automatic, it detects the distance. When a certain threshold of the distance is met, i.e. the car is present in front of the system, the software begins the image taking process. At this stage, the camera is synchronized with the Robotic Arm (discussed in the next section), it moves to different positions and takes images of the vehicle's body which are stored until the complete mapping of the bumper is complete. Finally, the images are processed using the image processing

software (discussed in later sections). In the case where the mode is set to manual, the software waits for the push of a button. Once the push button is pressed, an image is taken and it is processed through the same model through which the automatic images go. Once the image processing is complete, the data is saved into variables and then uploaded onto a spread sheet using the data logging module.

Data Logging Module

To implement the Data Logging module, we used the services of a Google API Console compatible with Google Drive. We start off by creating the credentials for our web server through which we access our spreadsheet data. This task generates a JSON file through Google which we place in the directory of our modules on the Raspberry Pi.

We then use the "oauth2client" and "gspread" libraries that are compatible with Python to achieve the desired task at hand. We access the spreadsheet through our web server and open the desired sheet on the file. Once that is complete, we simply write all the data saved through the image processing modules to the datasheet. An example of a generated dataset can be shown in Figure 12.

Model Number	Location	Scratch Detected	Percentage	Time	Mode	Car Number	Total Scratches
Toyota Corolla	Front Bumper	Not scratch	0.61829406	2018-04-11 15:24:34	manual	1	0
Toyota Corolla	Front Bumper	scratch	0.99999833	2018-04-11 15:47:19	manual	1	1
Toyota Corolla	Front Bumper	scratch	0.9999995	2018-04-11 15:48:03	manual	1	2
Toyota Corolla	Front Bumper	scratch	0.99999094	2018-04-11 15:49:04	manual	1	3
Toyota Corolla	Front Bumper	scratch	1	2018-04-11 15:50:24	manual	1	4
Toyota Corolla	Front Bumper	Not scratch	0.546876	2018-04-11 15:51:21	manual	1	4
Toyota Corolla	Front Bumper	scratch	0.83093405	2018-04-11 15:52:23	manual	1	5
Toyota Corolla	Front Bumper	Not scratch	0.90474826	2018-04-11 17:13:34	manual	1	5
Toyota Corolla	Front Bumper	Not scratch	0.9191349	2018-04-11 17:15:08	manual	1	5
Toyota Corolla	Front Bumper	Not scratch	0.92233723	2018-04-11 17:16:23	manual	1	5
Toyota Corolla	Front Bumper	scratch	0.99683136	2018-04-11 17:17:44	manual	1	6
Toyota Corolla	Front Bumper	Not scratch	0.85118735	2018-04-11 17:19:27	manual	1	6
Toyota Corolla	Front Bumper	Not scratch	0.8974405	2018-04-11 17:20:37	manual	1	6
Toyota Corolla	Front Bumper	scratch	0.9999691	2018-04-11 19:44:11	manual	1	7

Figure 12 – Scratch Detection Data

The Preprocessing module and the Trained Network module will be discussed in detail in the upcoming sections.

Device Casing

Once we finally had our device working, we moved onto toward designing a case for it in order to ensure portability and ease of use. We based our model on a small sized box with the distance sensor and the camera module facing forwards and the switches being available on the sides. The rest of the system was placed inside of the casing. A 3D model of the casing along with its dimensions can be shown in Figure 13.

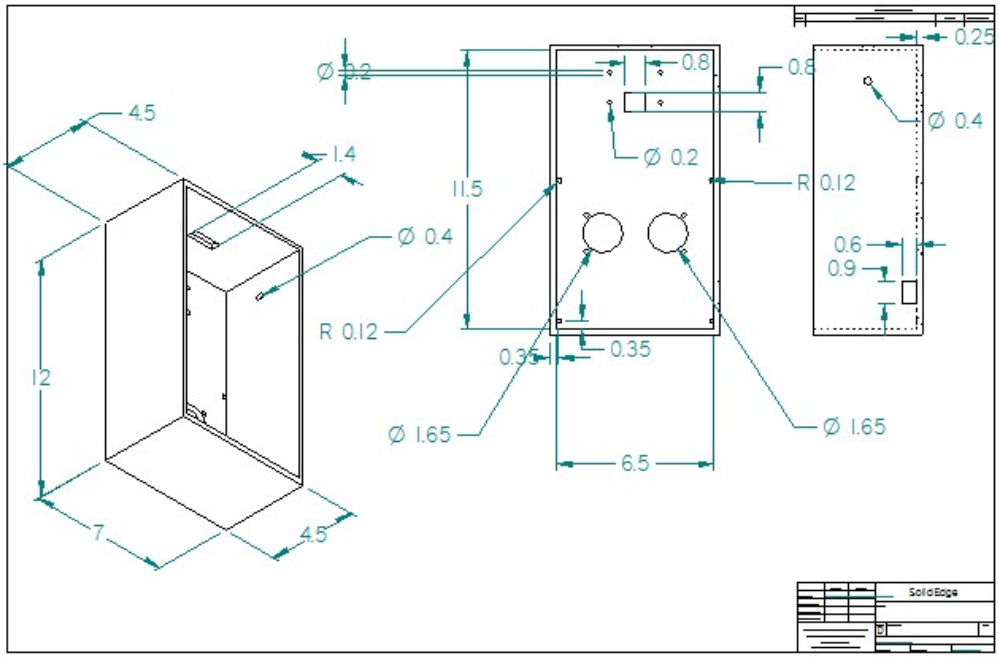


Figure 13 – 3D Model For Device Casing

Subsystem II – Image Processor

Research Methodology

The Image Processor Subsystem of our proposed solution serves as the core part of our implementation. The idea is to be able to develop a black box which takes input a captured image and outputs a decision stating whether it is a scratch or not.

In this section of our report, we aim to introduce our research methodology and how we went ahead with using our preprocessing techniques along with a Convolutional Neural Network (CNN) to achieve the task at hand. Our preprocessing methodology can be broken down into four independent steps, starting from region of interest extraction, artifact removal, and finally data augmentation

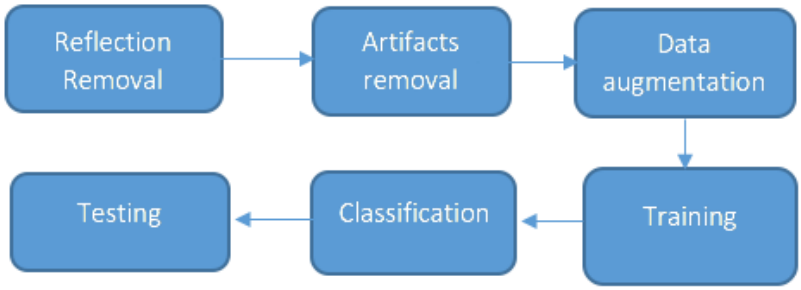


Figure 14 – Block Diagram Of Image Processing System

For training a machine learning model, we use Keras, which is a high level neural network API written in Python. It allows us to create Convolutional Neural Networks easily while changing our required parameters and training our data.

Environment

As part of our project, we require the training of a dataset on a machine learning model. The dataset contains images taken in a specific environment which constant variables. Due to this restriction we made our algorithm specific to only one environment, though it can be modifying easily for industry purposes. The main task was to remove all the unnecessary objects before we training the images. We use a cardboard to avoid reflection from the surface of vehicle before taking images to create a data set. A silver vehicle was used to create the data set. Images were taken during day time to capture detailed features from the vehicle surface.

The generated dataset contained a total of approximately 1500 images, split between scratches and non-scratches. This data can be better analyzed through Figure 15.

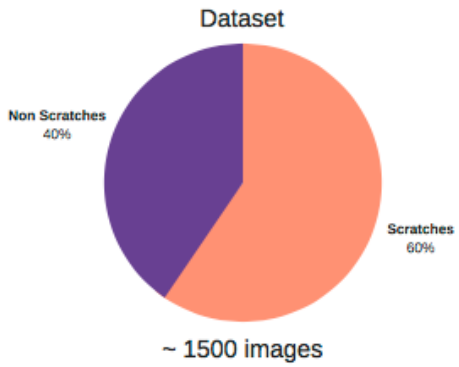


Figure 15 – Dataset Pie Chart

Preprocessing – Reflection Removal

Image Preprocessing is the name for operation on images at the lowest level of abstraction whose aim is to improve the image data that suppresses an undesired distortion in order to

produce better results. In our project we use Reflection removal, artifact removal and data augmentation as the image preprocessing techniques to improve accuracy.

When we create our data set we see large chunk of light being reflected from the surface of the car. However, we reduce that amount of reflection using card sheet, but still we need to improve the background reflection. For that we use Otsu thresholding in order to get the desired threshold of reflection. Which is followed me a set of Morphological operations (opening and dilations) to get the desired result.

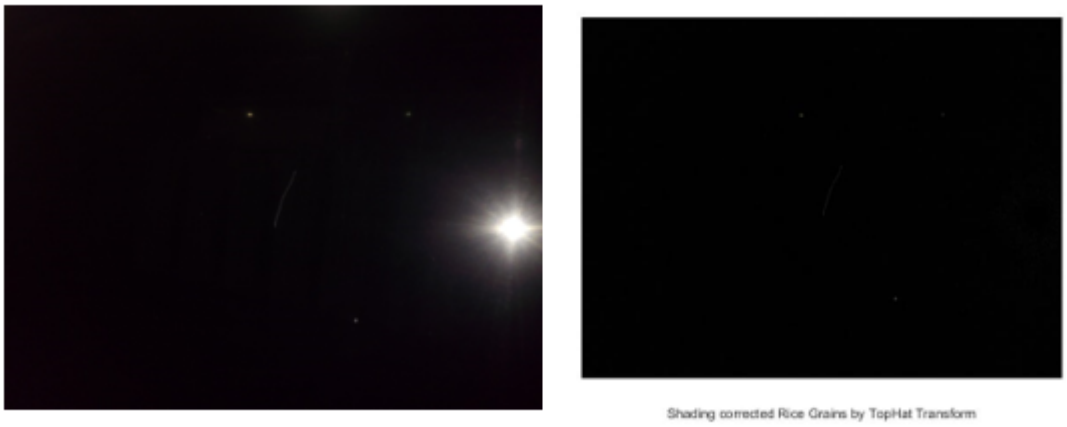


Figure 16 – Removal of Flash from Surface

Preprocessing – Artifacts Removal

Our project's main focus is the bumper of the vehicle. We need to eradicate all such objects that appear in the images that are not part of the front bumper's main body being analyzed. These include headlights, tires, etc. These need to be removed prior to training in order to ensure they are not interfering with our accuracy and results.

Using thresholding techniques, we implemented a simple program to remove any unwanted object present in the image aside from the surface of the bumper. Our results can be depicted in Figure 16. The image on the right is the original image, whereas the image of the left is one where the unwanted black object has been removed.

Figure 17 – Artifact Removal

Preprocessing – Data Augmentation

In spite of all the data available, fetching the right type of data which matches the exact use case of our result is a difficult task. Moreover, the data needs to be in good diversity as the object of interest need to be present in varying sizes, rotations, poses and lightning conditions.

To overcome this problem of limited quantity and diversity of data, we generate our own data with the existing data. Keras has ImageDataGenerator which can create data by using a single image and replicate the image by applying different factors, which include different rotation, width, height, shear, zoom and flip mode.

Figure 18 – A single image being augmented into 5 images

Introduction To Convolutional Neural Networks

Convolutional Neural Networks is a class of deep, feed forward artificial neural networks that has successfully been applied to analyzing visual imagery. They are inspired by biological processes which uses connectivity pattern between neurons resembles the organization of the animal visual cortex. CNN are made up of neurons that have learnable weights and biases. Each neuron receives an input after which it performs dot product and optionally followed by a nonlinear function.

Neural Network receives an input, and transform a series of hidden layers. Each hidden layer is made up of set of neurons, where each neuron is fully connected to all neurons in the previous layer. The last fully connected layer is called the output layer and in classification setting it represents the class scores.

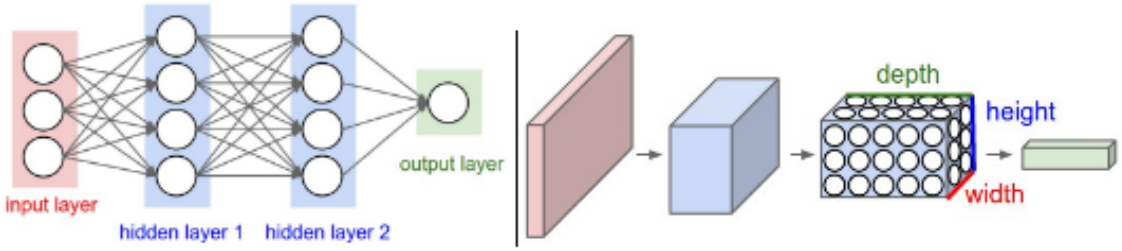


Figure 19 – Convolutional Neural Network Architecture

Training Our Model

Using the dataset we created earlier, we trained a Convolutional Neural Network which we implemented on Keras using Python. Initially, we began experimenting with the depth of our model to observe the accuracy of our results. We intended to come up with a model that would maximize our accuracy and minimize the losses.

Through our implemented model on Keras, we use 75% of the dataset for training data and the remaining 25% for validation data. We then trained our Convolutional Neural Network on a CPU with the dataset that was created using the preprocessing steps mentioned above.

An epoch is a parameter in the Convolutional Neural Network that is defined prior to training our model. An epoch is basically one forward and backward pass of all training images. For our model, we set the value to around 100 epochs given the size of our data in order to ensure our dataset is reviewed clearly.

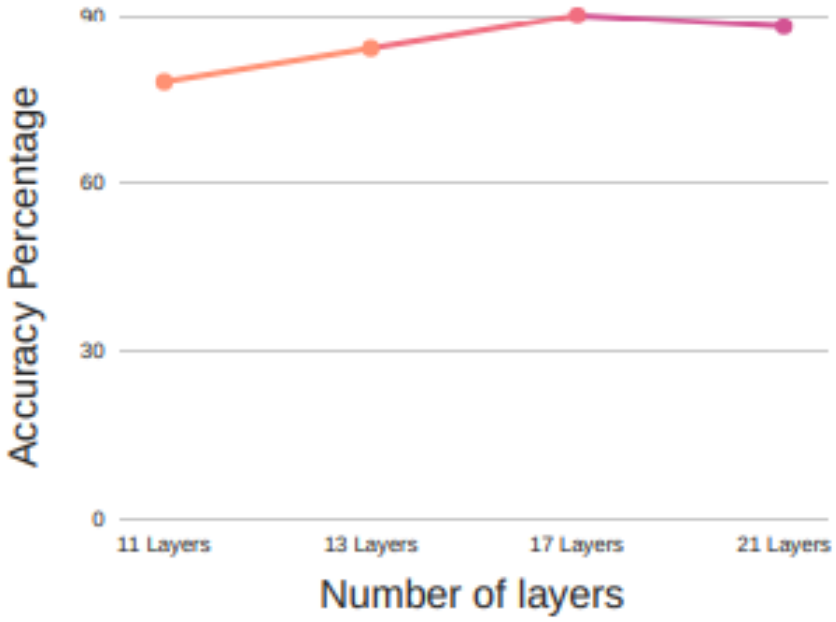


Figure 20 – Accuracy Of Model vs. Number of Layers

Figure 19 shows us the change in our model's prediction accuracy as the number of layers increased. Given our data, we noticed the highest accuracy at 17 layers. For this very reason, we set up our Convolutional Neural Network to be 17 layers deep. The choice of a design when it came to the Convolutional Neural Network allowed us to ensure maximum percentage accuracy to be generated on the model which would be used for testing images on the scratch detection device.

Our Convolutional Neural Network architecture can be visualized in Figure 21.

Finally, in order to classify our images between scratches and non-scratches, we modified our final layer of the neural network to output two classes in order to ensure the decision making process.

Now that our model was trained, it generated an accuracy of 90% on the given dataset. Moreover, a ".model" file was created which contains the model that can be used for testing purposes. We place this model on our scratch detection device and use it for classifying the images taken using the camera module.

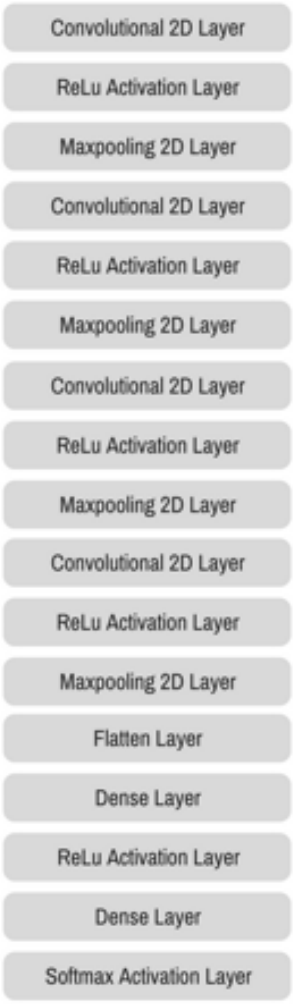


Figure 21 – Convolutional Neural Network (17 layers)

Eliminated Subsystem - Lighting Module

In our initial proposed solution, we had discussed the idea of a lighting module attached to our overall system which would help the camera capture more illuminated surfaces through the spreading of light at the right angles.

After careful inspection of the manufacturing process within the industry, we concluded that attaching the lighting module to our system would be infeasible. Instead, we propose that the lighting environment on the production line is improved to the order of minimizing the glare and reflection from the glossy surface of vehicles.

High luminance lights can cause glare and complicate visibility of objects. Hence to prevent such glares, the light source should be covered or partially obstructed. Number of recommendation are presented, firstly it can be done by correct positioning of luminaries. The light should not direct towards the lens of the camera when taking an image. Secondly to use large luminaries

with low luminance. Surface finishes that diffuse and scatter light should be used instead of glossy ones that create reflections. Lastly, luminaries with appropriate distribution of luminous density should be used.

The implementation of such a lighting system would only help our proposed system perform better and capture a larger number of scratches through the camera lens.

Prototype & System Implementation

In this section, we aim to discuss the working of our prototype in the context of the industrial implementation. Given that our problem is based in an environment which is continuously busy, we need to make certain assumptions in order to be able to propose a certain efficiency in terms of the implementation of the system. Our assumptions are as follows:

- The system will be implemented at a location at the end of the Trim Line and right before the beginning of the Chassis Line
- There will be minimal worker interference present in front of the system. At this point in the process, the workers will be working at a part of the vehicle that is away from the front bumper.
- The lighting conditions in the environment are going to be set up as described in the above sections.

In order to understand the optimal placement of our Automated Scratch Detection System, we conducted an analysis on the front bumper by modelling it mathematically to understand the number of images that need to be taken in relation with the distance, processing time, and accuracy of the system.

Mapping The Bumper

In order to understand the number of images that need to be taken on one vehicle, we began by experimenting with the field of view of our Raspberry Pi Camera Module V2. At different distances, we observed the size of the area that the camera captures. We then measured the dimensions of the bumper and calculated a rough area of the front bumper. This can be shown in Figure 21.

As soon as we had this information, we went ahead to observe the number of images that need to be taken in order for one system to map half of the front bumper given that it is placed at a certain distance from the vehicle.

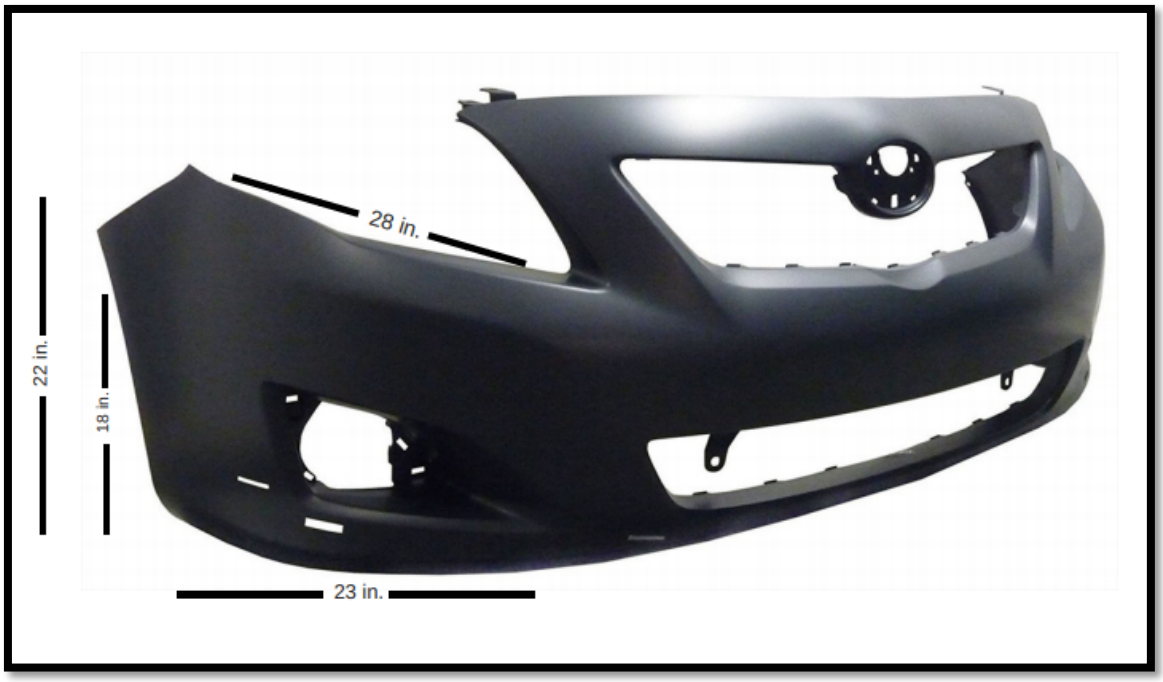


Figure 22 – Bumper Dimensions

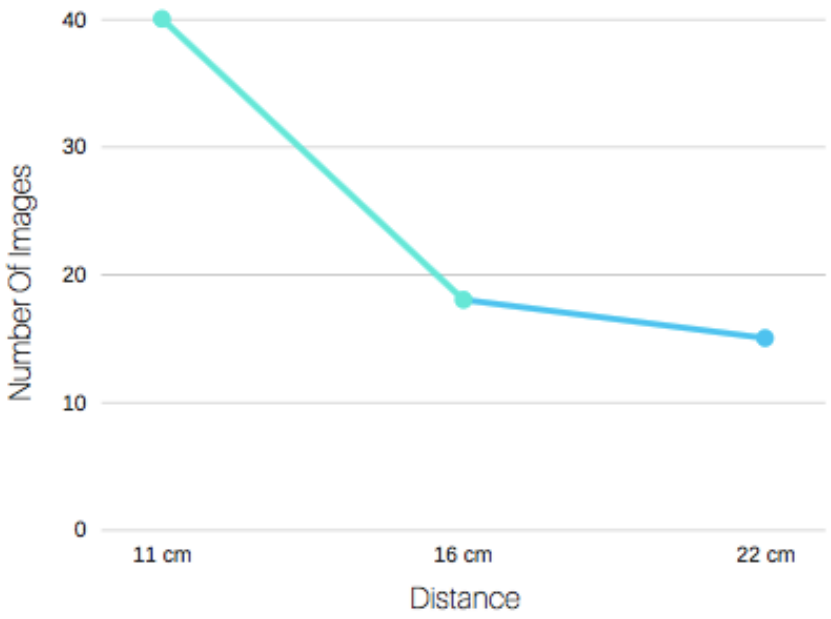


Figure 23 – Number Of Images vs. Distance

By performing a small experiment, we were able to understand that the number of images required to map the bumper decreases with increasing the distance of the placed system from the vehicle's body.

Given that one image requires a processing time of ~ 40 seconds on our Scratch Detection Device, the total processing time can be calculated. This relationship is depicted in the following Figure 24.

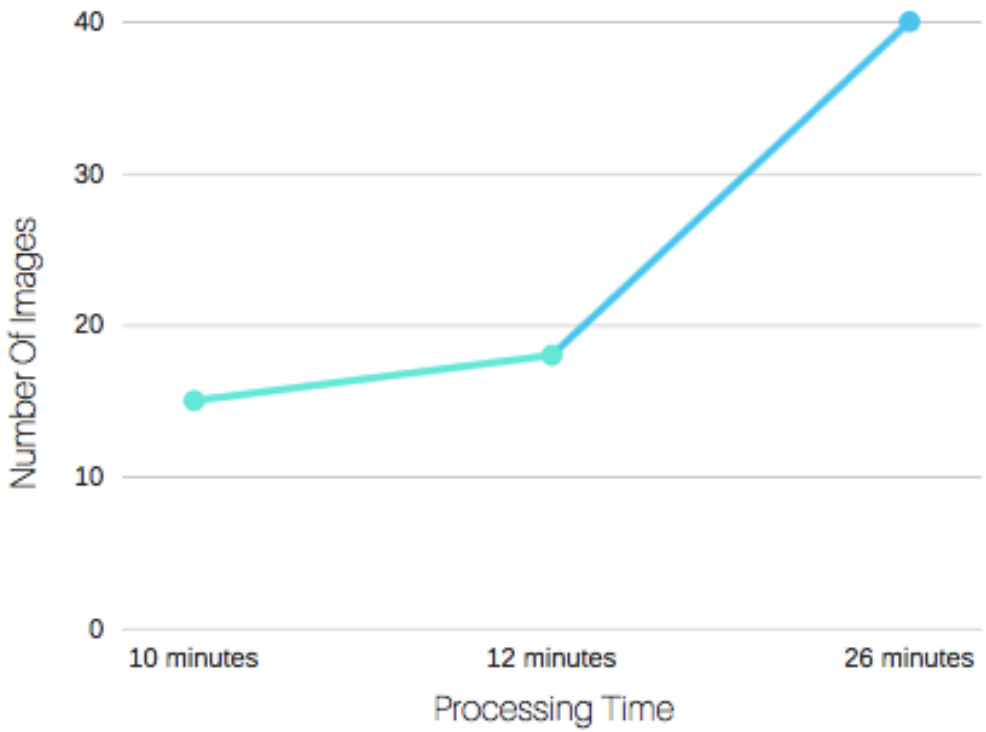


Figure 24 – Number of Images vs. Processing Time

From the graphical analysis shown above, we see that the highest number of images are required when the system is placed at 11 centimeters from the vehicle. However, the processing time for these images is also the highest, being around 26 minutes in total. Therefore, this makes the system inefficient in terms of its functioning with respect to time.

Further observations tell us that a distance of 16 centimeters would be optimal as the images taken would prove to have decent accuracy through our image processor, the processing time of 12 minutes would be manageable by the system in place.

Proposed Implementation

Keeping the abovementioned analysis in mind, we propose that two such automated scratch detection systems are put into place at a distance of 16 centimeters from the vehicle. This can be visualized in Figure 25.

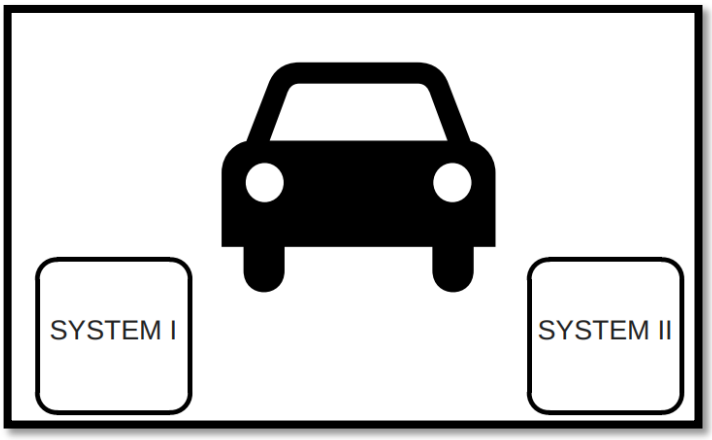


Figure 25– Proposed Implementation

Each of these system will map one half of the bumper, beginning from the side corners to the front midpoint of the bumper. These systems will be operating in parallel, therefore requiring a total time of 12 minutes to process the images taken of the whole bumper.

Test Cases & Results

We tested our prototype under different conditions and cases. We began by performing tests on the Robotic Arm to ensure all the degrees of freedom are working perfectly. Our Robotic Arm is able to map to the positions of a bumper through the use of the 6 installed motors. We performed these tests by inputting motor angle positions through the serial port on the Arduino Mega 2560. This allowed us to understand the range of the motors and finally know our mapping coordinates. We then created functions where the positions of the bumper where hardcoded, allowing the Robotic Arm to begin its mapping process once the distance threshold breaks.

We then set up the scratch detection device to check both its functioning as well a the image processor software installed on top of it. We initially began by testing the device using manual mode, where we took individual images of scratches and observed the results. We then set the device to automatic mode and synced our time delays with the robotic arm to ensure that the images are being taken as the robotic arm moves across the mapping of the bumper.
We compiled a data of around 60 test cases which we conducted on a vehicle using our system. Some of the cases can be seen in Figure 26.

Figure 26 – Test Cases

Using our tested data, we created a confusion matrix so that we are able to define some of our metrics to evaluate our performance on. The split between our 60 images can be seen using our confusion matrix shown in Figure 27.

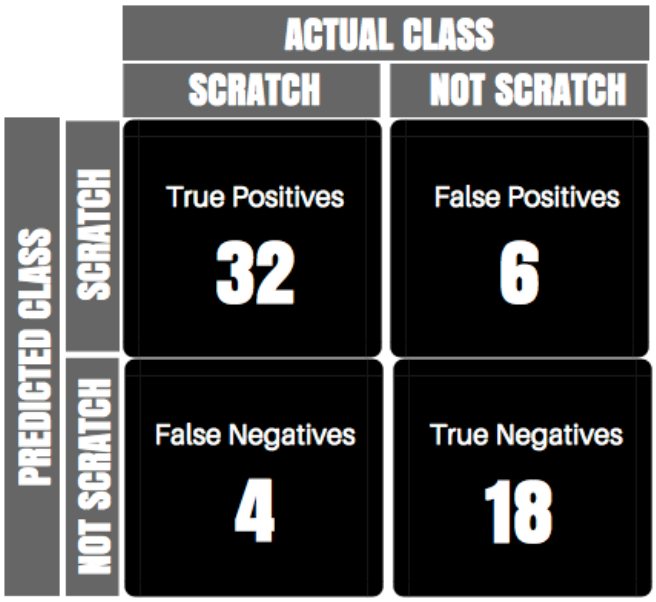


Figure 27 – Confusion Matrix

In order to better understand our performance, we chose to calculate accuracy, being the proportion of the total number of predictions that were made correctly. This can be done as follows:

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} = \frac{32 + 18}{32 + 6 + 4 + 18} = 83.33\%$$

We notice that our trained model provided us with an accuracy of 90% on the validation data that was set aside from the dataset. Our test cases provided us with an accuracy of 83.33%, showing us that the majority of the scratches present could be detected.

Scalability & Sustainability

The idea behind our proposed prototype was that it could potentially be up scaled to a larger industrial system that could cover the whole body of the vehicle. Given that two systems are in place, each could cover one half of the body if the reference point is set as a vertical line between the vehicle's body. If the industry were to find our prototype suitable, they could invest into a larger Robotic Arm which would function on the basis of our prototype to map the body of the vehicle and take images for processing them to check for scratches.

Moreover, our system has been designed to ensure user-friendliness. Keeping our two-state device in mind, users can simply toggle the switch for operation automatic mode and have the system do the rest of the work while it is running. In addition to that, they can simply detect the presence of a scratch by the click of a button. Our proposed system also makes the data logging process easier as the scratch data is now readily available online for the industry personnel to view.

Given that most industrial processes are now moving towards automation, our proposed system has been chosen keeping sustainability and future in mind. Automobile industries are moving towards automating each and every process involved in the manufacturing of the vehicle. Similarly, scratch detection will also serve as an area where the systems in place will be performing the tasks as opposed to human experts. The direction of development in this direction ensures that our solution is one that will not be obsolete in the upcoming years.

Future Prospects

Both the problem at hand and the proposed solution have plenty of room for development and improvement. The area we chose to work on has tremendous potential in the future. The integration of mechatronics systems with computer vision is what is driving industries towards progress these days. Keeping that in mind, the future prospects associated with this project are plenty.

Our focus has mainly been on the detection of scratches on the front bumper of a vehicle. There is room to expand to the remainder of the body while keeping in mind the different

artifacts that may be coming in the way during the scratch detection process i.e. windows, tires, etc.

While our machine learning model was trained on a dataset of a certain number of images taken of the front bumper. A larger dataset can be trained in order to detect the presence of scratches on the whole vehicle. The task at hand was quite challenging and being able to detect scratches whose characteristics fall in the range of millimeters is not easy.

It is necessary to be able to extract in-depth features of these images containing the vehicle's body while ensuring that the overfitting of data does not occur. A trained model which is capable of increasing accuracy and detection all possible scratches on a vehicle could be an end goal. This could be transformed into a problem statement in itself.

The current solution helps narrow down the responsibility and assists the trained experts in analyzing the scratches better. However, a potential end goal as a future prospect for a project like this could involve developing a system that independently is able to detect all possible scratches without the need for a manual system in place.

The future prospects of the Automated Scratch Detection System For The Automobile Industry can be summarized as follows:

- Improve the current machine learning model accuracy
- Upscale the Robotic Arm to cover the remainder of the vehicle's body
- Develop a more cost efficient solution to the problem at hand

Conclusion

The problem at hand revolved around developing an automated system which could narrow down the blame as to which point in the production line is responsible for the scratches as well as help the current inspection system improve. Our proposed solution aimed to do that through the implementation of a scratch detection device holding an image processing software interfaced with the robotic arm.

Our prototype has been successful in demonstrating the working of the Automated Scratch Detection System, we have been able to achieve desirable results and have tackled the problem at hand in a way to ensure that the problem statement's constraints and features are taken into account.

However, there is still room for improvement in our project. The dataset can be improved, which in turn will improve the trained model. This will result in a better accuracy when it comes to detection scratches. Given that the scratches we are dealing with are extremely small in size, it becomes difficult to capture them with the highest of efficiencies. However, the task at hand

was to help the current system in place and we were able to achieve that through our proposed solution.

References

[1] WonderTiger, “Robotic Arm: Arm by WonderTiger,” Thingiverse. [Online]. Available: https://www.thingiverse.com/thing:1838120. [Accessed: 24-Apr-2018].

[2] Sonka, M., Hlavac, V., & Boyle, R. (2015). *Image processing, analysis, and machine vision.*

[3] Al-Jabbouli, Hasan & Koç, Ebubekir. (2015). Scratches Detection in Extruded Aluminium Profiles Using Image Processing. Joussrnal of Engineering Science and Technology. 7. 221-225.

[4] Arun Mohan, Sumathi Poobal, Crack detection using image processing: A critical review and analysis, In Alexandria Engineering Journal, 2017, , ISSN 1110-0168, https://doi.org/10.1016/j.aej.2017.01.020 .
(http://www.sciencedirect.com/science/article/pii/S1110016817300236)

[5] Sergei Gontscharov, Hauke Baumgärtel, Andre Kneifel, Karl-Ludwig Krieger, Algorithm Development for Minor Damage Identification in Vehicle Bodies Using Adaptive Sensor Data Processing, In Procedia Technology, Volume 15, 2014, Pages 586-594, ISSN 2212-0173, https://doi.org/10.1016/j.protcy.2014.09.019 .
(http://www.sciencedirect.com/science/article/pii/S2212017314001340)

[6] High-resolution camera help Hailscanner detect every dent. (2016, September 13). Retrieved January 03, 2018, from https://www.alliedvision.com/en/news/detail/news/hail-damage-high-resolution-camerasdetect-every-dent.html

[7] Bourgeat, P., & Meriaudeau, F. (2002). . *Machine Vision Applications in Industrial Inspection X* . doi:10.1117/12.460196

[8] Baumgaertel, Klaas & Skwarek, Volker & Kneifel, A & Gontscharov, Sergei & Krieger, Karl-Ludwig. (2014). Detecting car damages with CAN networks. CAN in Automation Newsletter. 1. 38-40.

[9] Fábio Celestino Pereira, Carlos Eduardo Pereira, Embedded Image Processing Systems for Automatic Recognition of Cracks using UAVs, In IFAC-PapersOnLine, Volume 48, Issue 10, 2015, Pages 16-21, ISSN 2405-8963, https://doi.org/10.1016/j.ifacol.2015.08.101.

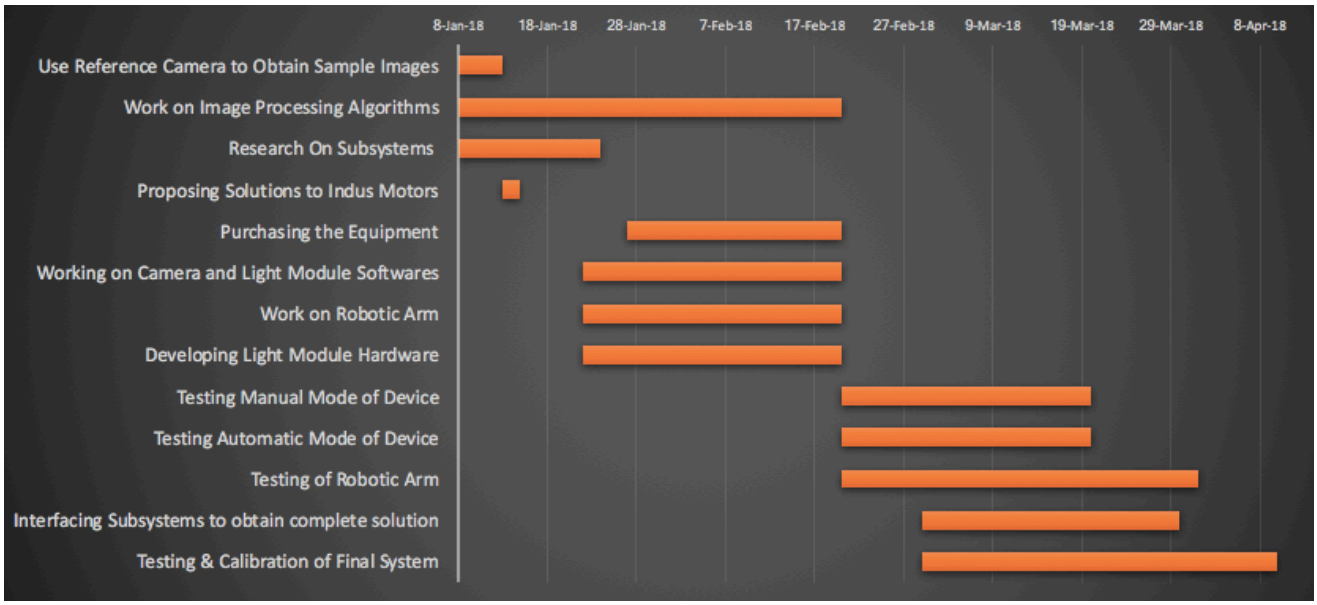
[10] Forsyth, D., & Ponce, J. (2015). *Computer vision: a modern approach* . Boston: Pearson.ma

Appendix

A-1 – Budget

S.No	Name of Component	Quantity	Rate (Rs.)	Total Amount (Rs.)
1	MG 945 Servo motor	7	550	3850
2	Stepper Motor	1	300	300
3	Jumper wires M+M	1	160	160
4	Accrylic sheet	1	220	220
5	Raspberry Pi camera Module V2	1	2925	2925
6	Switch	3	5	15
7	Reset Button	3	2	6
8	USB Cabel	1	50	50
9	Ultrasonic Distance Sensor	2	150	300
10	Acrylic material Robotic arm	4	4500	4500
11	Acrylic material Device	1	1200	1200
Total				13526

A-2 – Gantt Chart



A-3 – Software

Robotic Arm Code

```
#include <VarSpeedServo.h>
#include <Stepper.h>

#define echoPin 12
#define trigPin  13
// create servo objects
VarSpeedServo myservo1;
VarSpeedServo myservo2;
VarSpeedServo myservo3;
VarSpeedServo myservo4;
VarSpeedServo myservo5;
VarSpeedServo myservo6;

const int stepsPerRevolution = 200; // change this to fit the number of steps per revolution for
your motor
// initialize the stepper library on pins 8 through 11:
Stepper myStepper(stepsPerRevolution, 8, 9, 10, 2);
long duration;

void setup() {
  int z,o,x,y;

 Serial.begin(9600);
 pinMode(trigPin, OUTPUT);
 pinMode(echoPin, INPUT);
 // set the speed at 60 rpm:
 myStepper.setSpeed(60);
 // initialize the serial port:
 Serial.begin(9600);

 myservo1.attach(4); //base motor
 myservo2.attach(5); //base motor
 myservo3.attach(6); //lower motor
 myservo4.attach(7); //upper motor
 myservo5.attach(3);//upper motor

 z = distanceSensor();

 basemotors(120,50);
 lowermotors(150,50);
 uppermotors(0, 50);
 clampmotors(120,50);
}
```

```
void loop() {
  int distance;
  unsigned long t;

  distance = distanceSensor();
  Serial.print("ditance = ");Serial.println(distance,DEC);

  if (distance < 4){
     t = millis();

     delay(2000);

     motorPosition(200,120,100,0,160);
     delay(4000);
     motorPosition(200,120,120, 30, 160);
     delay(4000);
     motorPosition(-500,90,150,0,120);
     delay(4000);
     motorPosition(-100,120,90,0,70);


    while (millis()- t < 50000) {
       delay(1000);
     }
    stepper(900);
    }
  }


int stepper(int revolutions){
   myStepper.step(revolutions);
}

int basemotors(int angle, int speed){
   myservo1.write(angle, speed, false);
   myservo2.write(180 - angle ,speed, false);
}

int lowermotors(int angle,int speed){
   myservo3.write(angle,speed);
}

int uppermotors(int angle,int speed){
   myservo4.write(angle,speed);
```

```
}

int clampmotors(int angle,int speed){
   myservo5.write(angle,speed);
}

int distanceSensor(){
   long distance;

   digitalWrite(trigPin, HIGH);
   delayMicroseconds(1000); //- Removed this line

   digitalWrite(trigPin, LOW);
   duration = pulseIn(echoPin, HIGH);

   distance = (duration/2) / 29.1;

   return distance;
}

int normalPosition() {
   basemotors(120,50);
   lowermotors(90,50);
   uppermotors(10, 50);
   clampmotors(160,50);
}

int backPositions(int clampServo,int upperServo, int lowerServo, int baseServo){
    lowermotors(lowerServo,50);
    basemotors(baseServo,50);
    uppermotors(upperServo,50);
    clampmotors(clampServo,50);
}

int motorPosition(int s,int baseServo,int lowerServo,int upperServo,int clampServo){
   stepper(s);
   basemotors(baseServo,50);
   lowermotors(lowerServo,50);
   uppermotors(upperServo,50);
   clampmotors(clampServo,50);
}
```

Scratch Detection Device – dataAquisition.py

```
import RPi.GPIO as GPIO
import time
import datetime
import picamera

TRIG = 18
ECHO = 22
MODE_AUTO = 11
MODE_MANUAL = 37

#intiliazing camera
camera = picamera.PiCamera()

def pinSetup():
        GPIO.setwarnings(False)
        GPIO.setmode(GPIO.BOARD)

        GPIO.setup(TRIG,GPIO.OUT)
        GPIO.setup(ECHO,GPIO.IN)
        GPIO.setup(MODE_AUTO,GPIO.IN, pull_up_down = GPIO.PUD_DOWN)
        GPIO.setup(MODE_MANUAL,GPIO.IN, pull_up_down = GPIO.PUD_DOWN)

        print("Setup Complete");

def ultrasonicSetup():
   GPIO.output(TRIG, False)
   time.sleep(2)

def getDistance():
        GPIO.output(TRIG, True)
        time.sleep(0.00001)
        GPIO.output(TRIG, False)

        pulse_start = 0;
        pulse_end = 0;
        pulse_duration = 0;

        while GPIO.input(ECHO)==0:
        pulse_start = time.time()

                if GPIO.input(MODE_AUTO) == 0:
                        break

        while GPIO.input(ECHO)==1:
```

```
        pulse_end = time.time()

        pulse_duration = pulse_end - pulse_start

        distance = pulse_duration * 17150

        distance = round(distance, 2)

        return distance

def detectMode():
        if GPIO.input(MODE_MANUAL) == 1 and GPIO.input(MODE_AUTO) == 0:
                return "manual"

        elif GPIO.input(MODE_AUTO) == 1 and GPIO.input(MODE_MANUAL) == 0:
        return "automatic"

def labelImage():
        now = datetime.datetime.now()
        label = "Image" + str(now) + ".jpg"

        return label

def captureImage(label):
        camera.vflip = True
        camera.hflip = True

        camera.capture(label)
```

Scratch Detection Device – interface.py

```
import dataAquisition
from dataAquisition import *

dataAquisition.pinSetup()
dataAquisition.ultrasonicSetup()

import test_network
import dataLogging
import time

model = "detectionModel"

while True:
```

```
imageLabels = []

    while True:
            mode = dataAquisition.detectMode()

            print(mode)

            if mode == 'automatic':
                    dist = dataAquisition.getDistance()
                    print("Distance: " + str(dist) + "cm")

                    if dist < 5 and dist > 0:
                            time.sleep(4)
                    for i in range(4):
                    label = dataAquisition.labelImage()
                    dataAquisition.captureImage(label)
                    imageLabels.append(label)
                    print (imageLabels)
                                    time.sleep(5)
                    break

            elif mode == 'manual':
                    if GPIO.input(MODE_MANUAL) == 1:
                       print("Manual Operation")
                       label = dataAquisition.labelImage()
                       dataAquisition.captureImage(label)
                       break

            elif mode == 'None':
                    break

     if mode == 'automatic':
       for image in imageLabels:
         info = test_network.testImage(image, model)
         dataLogging.loadData(info[0], info[1], mode)

elif mode == 'manual':
        info = test_network.testImage(label, model)
        dataLogging.loadData(info[0], info[1], mode)
```

Scratch Detection Device – Preprocessor

```
import cv2 as cv
import numpy as np
```

```
from matplotlib import pyplot as plt

#Reflection Removed
img = cv.imread('image.jpg',0)
ret,thresh1 = cv.threshold(img,127,255,cv.THRESH_BINARY)

opening = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)

dilation = cv2.dilate(img,kernel,iterations = 1)

#Artifact Removal

height, width = img.shape

for row in range(height-1):
   for col in range(width-1):
      img[row, col] = 255 if img[row,col] <= 70 else 0

cv2.imwrite('outputNew.jpg',imgTwo)
```

Scratch Detection Device – test_network.py

```
# import the necessary packages
from keras.preprocessing.image import img_to_array
from keras.models import load_model
import numpy as np
import argparse
import imutils
import cv2
import os
import datetime

def testImage(image, model):
# load the image
        image = cv2.imread(image)
        orig = image.copy()

# pre-process the image for classification
        image = cv2.resize(image, (28, 28))
        image = image.astype("float") / 255.0
        image = img_to_array(image)
        image = np.expand_dims(image, axis=0)

# load the trained convolutional neural network
```

```
        print("[INFO] loading network...")
        model = load_model(model)

# classify the input image
        (notscratch, scratch) = model.predict(image)[0]

# build the label
        label = "scratch" if scratch > notscratch else "Not scratch"
        proba = scratch if scratch > notscratch else notscratch
        info = "{}: {:.2f}%".format(label, proba * 100)

# draw the label on the image
        output = imutils.resize(orig, width=400)
        cv2.putText(output, label, (10, 25),  cv2.FONT_HERSHEY_SIMPLEX,
                0.7, (0, 255, 0), 2)

# show the output image
        now = datetime.datetime.now()
        cv2.imwrite('Test_Output_' + str(now) + '.jpg', output)

        information = [label, str(proba)]

        return information
```

Scratch Detection System – dataLogging.py

```
import json
import gspread
import oauth2client.client
from oauth2client.service_account import ServiceAccountCredentials
import datetime

JSON_FILENAME = 'scratch_detection_data.json'
GSHEET_NAME = 'Scratch Detection Data'
SCOPES = ['https://spreadsheets.google.com/feeds', 'https://www.googleapis.com/auth/drive']

def loadData(label, probability, mode):

        json_key = json.load(open(JSON_FILENAME))
        credentials = ServiceAccountCredentials.from_json_keyfile_name(JSON_FILENAME,
SCOPES)

        file = gspread.authorize(credentials)
```

```
        sheet = file.open(GSHEET_NAME).sheet1

        row = 1
        col = 1

        while (sheet.cell(row, col).value) != '':
                row = row + 1

        now = datetime.datetime.now()

        sheet.update_cell(row,1,"Toyota Corolla")
        sheet.update_cell(row,2,"Front Bumper")
        sheet.update_cell(row,3, label)
        sheet.update_cell(row,4, probability)
        sheet.update_cell(row,5, str(now))
        sheet.update_cell(row,6, mode)
```

Image Processor – lenet.py

```
# import the necessary packages
from keras.models import Sequential
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.layers.core import Activation
from keras.layers.core import Flatten
from keras.layers.core import Dense
from keras import backend as K

class LeNet:
        @staticmethod
        def build(width, height, depth, classes):
                # initialize the model
                model = Sequential()
                inputShape = (height, width, depth)

                # if we are using "channels first", update the input shape
                if K.image_data_format() == "channels_first":
                        inputShape = (depth, height, width)

                # first set of CONV => RELU => POOL layers
                model.add(Conv2D(20, (5, 5), padding="same",
                        input_shape=inputShape))
                model.add(Activation("relu"))
                model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))
```

```
        # second set of CONV => RELU => POOL layers
        model.add(Conv2D(50, (5, 5), padding="same"))
        model.add(Activation("relu"))
        model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

        # third set of CONV => RELU => POOL layers
        model.add(Conv2D(50, (5, 5), padding="same"))
        model.add(Activation("relu"))
        model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

        # fourth set of CONV => RELU => POOL layers
        model.add(Conv2D(50, (5, 5), padding="same"))
        model.add(Activation("relu"))
        model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

        # first (and only) set of FC => RELU layers
        model.add(Flatten())
        model.add(Dense(500))
        model.add(Activation("relu"))

        # softmax classifier
        model.add(Dense(classes))
        model.add(Activation("softmax"))

        # return the constructed network architecture
        return model
```

Image Processor – train_network.py

```
# USAGE
# python train_network.py --dataset images --model crack_not_crack.model

# set the matplotlib backend so figures can be saved in the background
import matplotlib


# import the necessary packages
from keras.preprocessing.image import ImageDataGenerator
from keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from keras.preprocessing.image import img_to_array
from keras.utils import to_categorical
from pyimagesearch.lenet import LeNet
```

```
from imutils import paths
import matplotlib.pyplot as plt
import numpy as np
import argparse
import random
import cv2
import os

matplotlib.use("Agg")

# construct the argument parse and parse the arguments
ap = argparse.ArgumentParser()
ap.add_argument("-d", "--dataset", required=True,
	help="path to input dataset")
ap.add_argument("-m", "--model", required=True,
	help="path to output model")
ap.add_argument("-p", "--plot", type=str, default="plot.png",
	help="path to output loss/accuracy plot")
args = vars(ap.parse_args())

# initialize the number of epochs to train for, initia learning rate,
# and batch size
EPOCHS = 100
INIT_LR = 1e-3
BS = 32

# initialize the data and labels
print("[INFO] loading images...")
data = []
labels = []

# grab the image paths and randomly shuffle them
imagePaths = sorted(list(paths.list_images(args["dataset"])))
random.seed(42)
random.shuffle(imagePaths)

# loop over the input images
for imagePath in imagePaths:
	# load the image, pre-process it, and store it in the data list
	image = cv2.imread(imagePath)
	print(imagePath)
	image = cv2.resize(image, (28, 28))
	image = img_to_array(image)
	data.append(image)
```

```
        # extract the class label from the image path and update the
        # labels list
        label = imagePath.split(os.path.sep)[-2]
        label = 1 if label == "scratch" else 0
        labels.append(label)

# scale the raw pixel intensities to the range [0, 1]
data = np.array(data, dtype="float") / 255.0
labels = np.array(labels)

# partition the data into training and testing splits using 75% of
# the data for training and the remaining 25% for testing
(trainX, testX, trainY, testY) = train_test_split(data,
        labels, test_size=0.25, random_state=42)

# convert the labels from integers to vectors
trainY = to_categorical(trainY, num_classes=2)
testY = to_categorical(testY, num_classes=2)

# construct the image generator for data augmentation
aug = ImageDataGenerator(rotation_range=30, width_shift_range=0.1,
        height_shift_range=0.1, shear_range=0.2, zoom_range=0.2,
        horizontal_flip=True, fill_mode="nearest")

# initialize the model
print("[INFO] compiling model...")
model = LeNet.build(width=28, height=28, depth=3, classes=2)
opt = Adam(lr=INIT_LR, decay=INIT_LR / EPOCHS)
model.compile(loss="binary_crossentropy", optimizer=opt,
        metrics=["accuracy"])

# train the network
print("[INFO] training network...")
H = model.fit_generator(aug.flow(trainX, trainY, batch_size=BS),
        validation_data=(testX, testY), steps_per_epoch=len(trainX) // BS,
        epochs=EPOCHS, verbose=1)

# save the model to disk
print("[INFO] serializing network...")
model.save(args["model"])

# plot the training loss and accuracy
plt.style.use("ggplot")
```

```
plt.figure()
N = EPOCHS
plt.plot(np.arange(0, N), H.history["loss"], label="train_loss")
plt.plot(np.arange(0, N), H.history["val_loss"], label="val_loss")
plt.plot(np.arange(0, N), H.history["acc"], label="train_acc")
plt.plot(np.arange(0, N), H.history["val_acc"], label="val_acc")
plt.title("Training Loss and Accuracy on crack/Not crack")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="lower left")
plt.savefig(args["plot"])
```