Car Damage Analysis

Technical Manual - David Weir (Student No. 15432642)

Technical Manual

Project Name:	Car Damage Analysis
Author:	David Weir
Student No.	15432642
Supervisor:	Dr. Suzanne Little
Date:	18/05/19

Abstract:

Car Damage analysis is a project which aims to streamline the returns process in car rental companies at the moment.

Images of cars are captured by an enclosure surrounding the car, these images are then analysed using machine learning algorithms in order for a prediction to be made as to whether the car is damaged or not. Damaged cars are displayed to employees through a web application.

This project aids in increasing the productivity of employees in car rentals companies by reducing the amount of time taken to analyse a car for damages.

1 Introduction

1.1 Overview

Currently at the end of a car rental, each car is manually inspected by teams of employees on a constant rota. This system aims to automate this process, allowing for these employees to increase productivity by minimising the number of cars which need to be manually evaluated. In order to minimise the number of cars to be evaluated, every car is scanned by this system and is deemed to be either damaged or undamaged. Employees will then only be required to assess the cars which have been categorised as damaged.

In this system, a car will enter an enclosure where its car registration will be read and images will be taken. These images are then sent to be analysed to check whether or not they contain damage. Should an image be classified as damaged, it is then sent to the application's web app, where an employee can inspect the images in order to appraise the car's damages.

This report outlines the various design decisions involved with this system, why these decisions were made and ultimately how these designs were implemented into a functional system. Some of the problems and resolutions uncovered during this project will also be discussed. An installation guide will outline how this system can be deployed for use in a car rental company. Finally, some of the possible areas of future work associated with this project will be discussed.

1.2 Glossary

- **Appraisal:** The action taken by an employee of a rental car company in which they appraise the damage of a car in order to generate a bill for a customer. More specifically in this system, an appraisal contains images which can be used by an employee while appraising damages.
- **Enclosure:** The hardware portion of this project is comprised of two cameras and a Raspberry Pi computer, which detect car registrations and capture images of the car in order for them to be analysed for damage.
- **SVM:** Support vector machine a supervised learning model, which is used for classification. The SVM in this project is trained to detect damage in an image of a car.
- **HOG:** Histogram of oriented gradients is a feature descriptor most commonly used in computer vision to perform object detection.
- **SURF:** speeded up robust features, a patented local feature detector which is commonly used in object recognition.
- **OCR:** Optical character recognition is the detection of characters from an image using computer software. This project makes use of OCR when detecting car registration plates.
- **OpenCV:** A library of functions which are used in computer vision tasks, a number of versions are available, this project uses the Python version.
- **Model Training:** Providing a machine learning algorithm with pre-processed images/data for it to learn from.

1.3 Research

There has been some research into the detection of damage in images of cars in the past, such as a report published by the University of Amsterdam and PwC in 2018 who achieved an accuracy of 90% when detecting damage in images of cars. Before beginning work on the project, I had to complete a number of research steps.

- de Deijn, J. (2018). Vrije Universiteit Amsterdam. Automatic Car Damage Recognition using Convolutional Neural Networks.

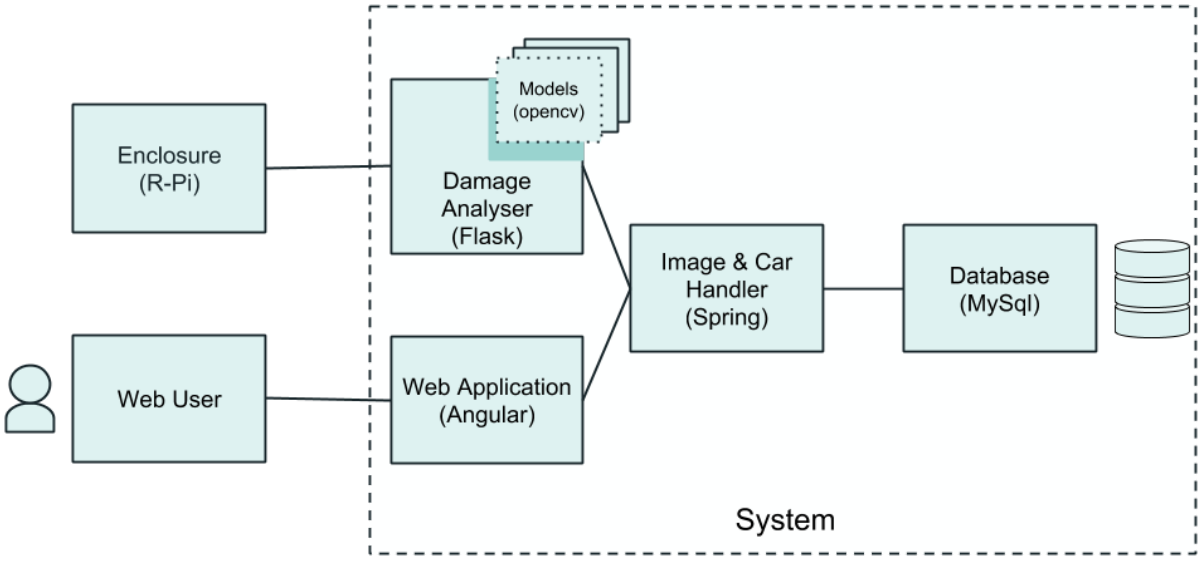
After conversations with employees from these companies, I had to define what exactly I could classify as damage. I decided to focus on images of larger dents and scrapes on cars, due to the fair wear and tear policies in place in rental companies. Damage such as cracked wing mirrors and broken headlights are out of scope for this project.

I investigated online to find an available dataset which I could use for training a machine learning model, but found there to be very few readily available online. This lead to the realisation that I would need to create my own dataset. I found that there was a large general car dataset available from Stanford which I would be able to clean and supplement to create the dataset I required.

Due to the lack of data available to me to solve the problem, I decided not to develop the solution in a similar manner to that of the report from University of Amsterdam, as they used a convolutional neural network which requires a huge amount of data. The use of machine learning algorithms rather than deep learning models allowed me to have a hand in the extraction of features from images, and thus required that I collect less data.

A large portion of my allocated time was spent learning about how various machine learning and computer vision algorithms work in the first semester. I used this time to choose algorithms which I felt might be able to solve the problem from detecting images.

2 System Architecture



A car enters the enclosure, which is powered by a Raspberry Pi, where its registration plate is retrieved from a video stream using OpenCV and Google's Tesseract OCR engine. When a registration plate is detected, multiple images of the car are captured and sent to be analysed by the damage analyser via a HTTP POST request.

The POST request from the enclosure is handled using flask, written in Python. When the images have been retrieved, they are predicted for damages using a HOG Descriptor SVM model, written using Python OpenCV This has been trained to classify images of cars into damaged or undamaged categories. As can be seen in the diagram, the analyser was written to allow new models for categorisation to be implemented with greater ease, allowing me to continue prototyping new methods of delivering the damage analyser model.

The damaged images are then sent to the Image & Car Handler via HTTP call which is handled by Spring, written in Java. The images are then stored in the local filesystem, and a reference to the image is stored in the database. References to the images are stored in order to increase the speed of the database as storing the images as binary can cause the database to slow. Entries are created in the database for a new appraisal of the damaged car images.

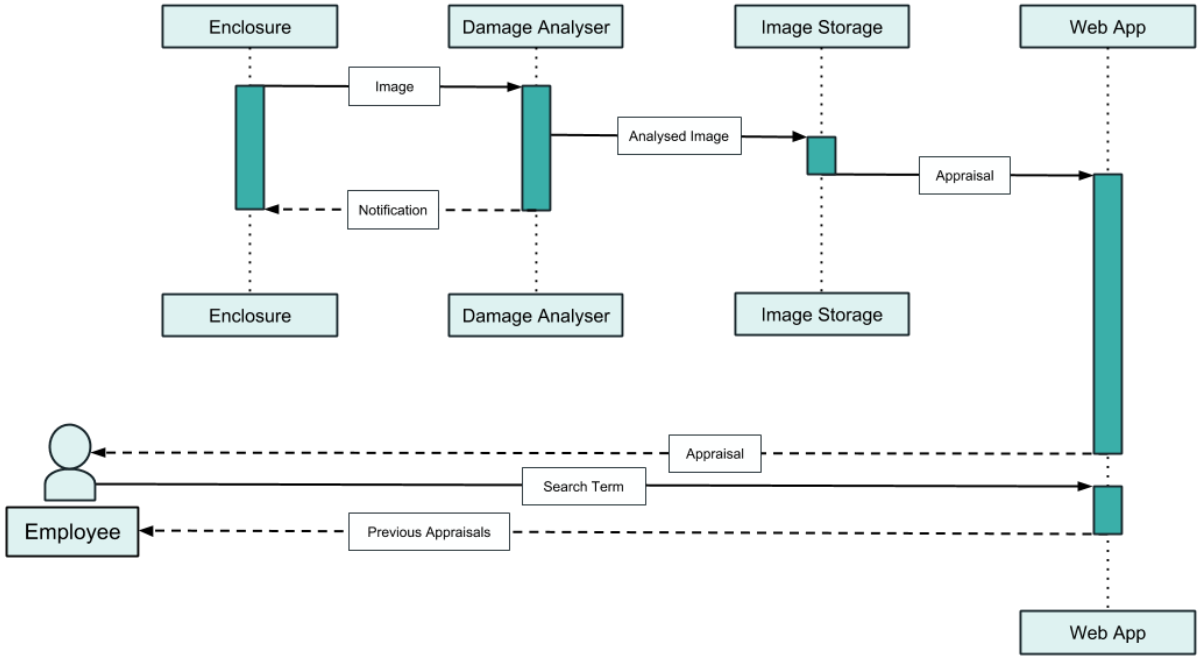
The main users of the application - rental company employees - will interact with the system through the use of a web application developed using Angular 7. The main pages of the application are populated by Angular via HTTP GET requests to the Image & Car Handler, which return images, car registrations, and rental information. The application allows employees to view cars for appraisals, mark appraisals as complete, search for previously appraised cars by car registration, and add new cars to the database.

3 Design

The following sections will provide an overview of the system as a whole. This will be achieved through the use of various high level design diagrams, and additional diagrams to showcase some of the other design decisions made while developing the system.

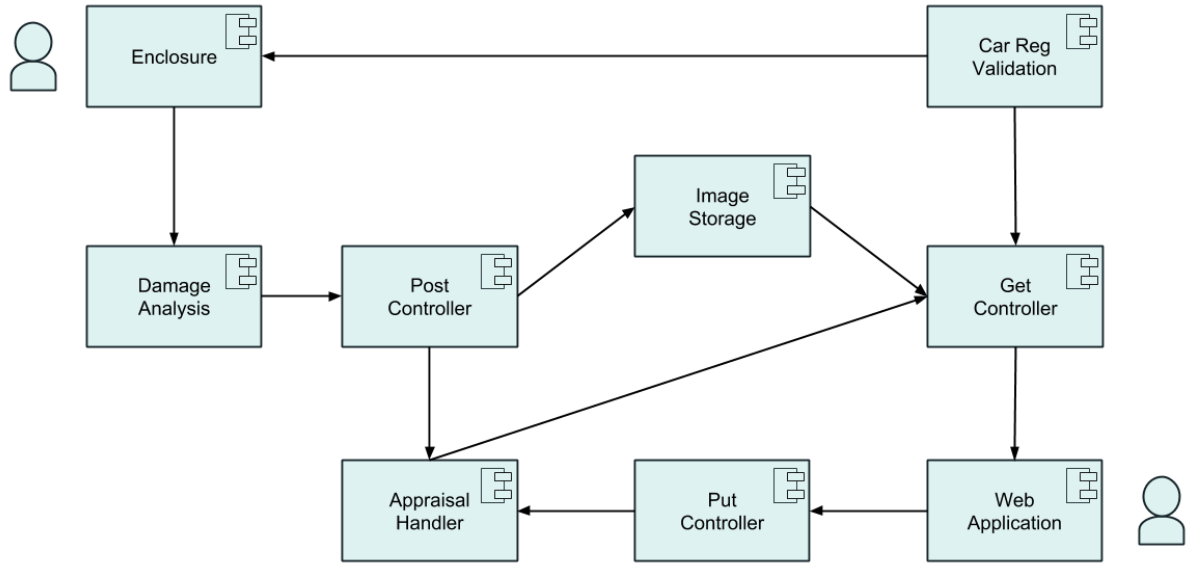
3.1 High Level Design

3.1.1 Sequence Diagram

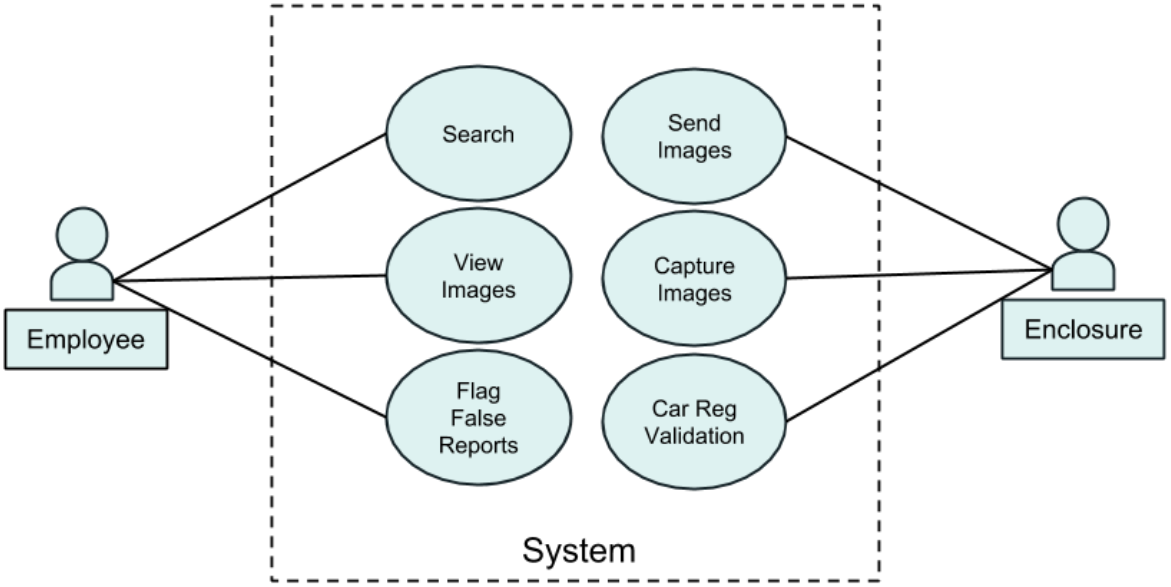


The sequence diagram produces a visual walkthrough of the sequence of steps which are undertaken when creating and completing an appraisal in this system. Unlike any of the other high level design diagrams, this sequence diagram purposefully displays the order in which certain functionality is executed from beginning to end.

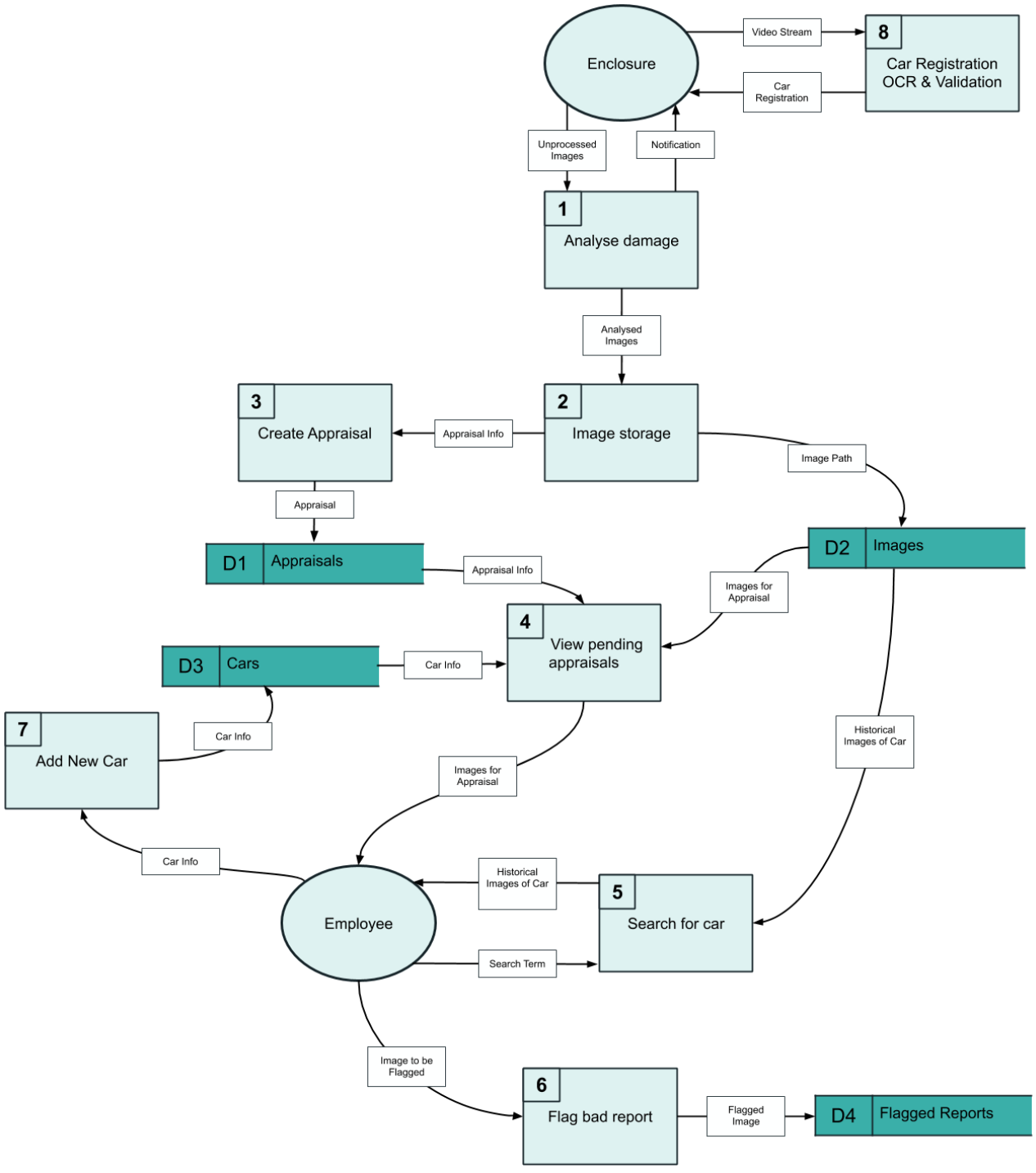
3.1.2 Component Diagram



3.1.3 Use Case Diagram

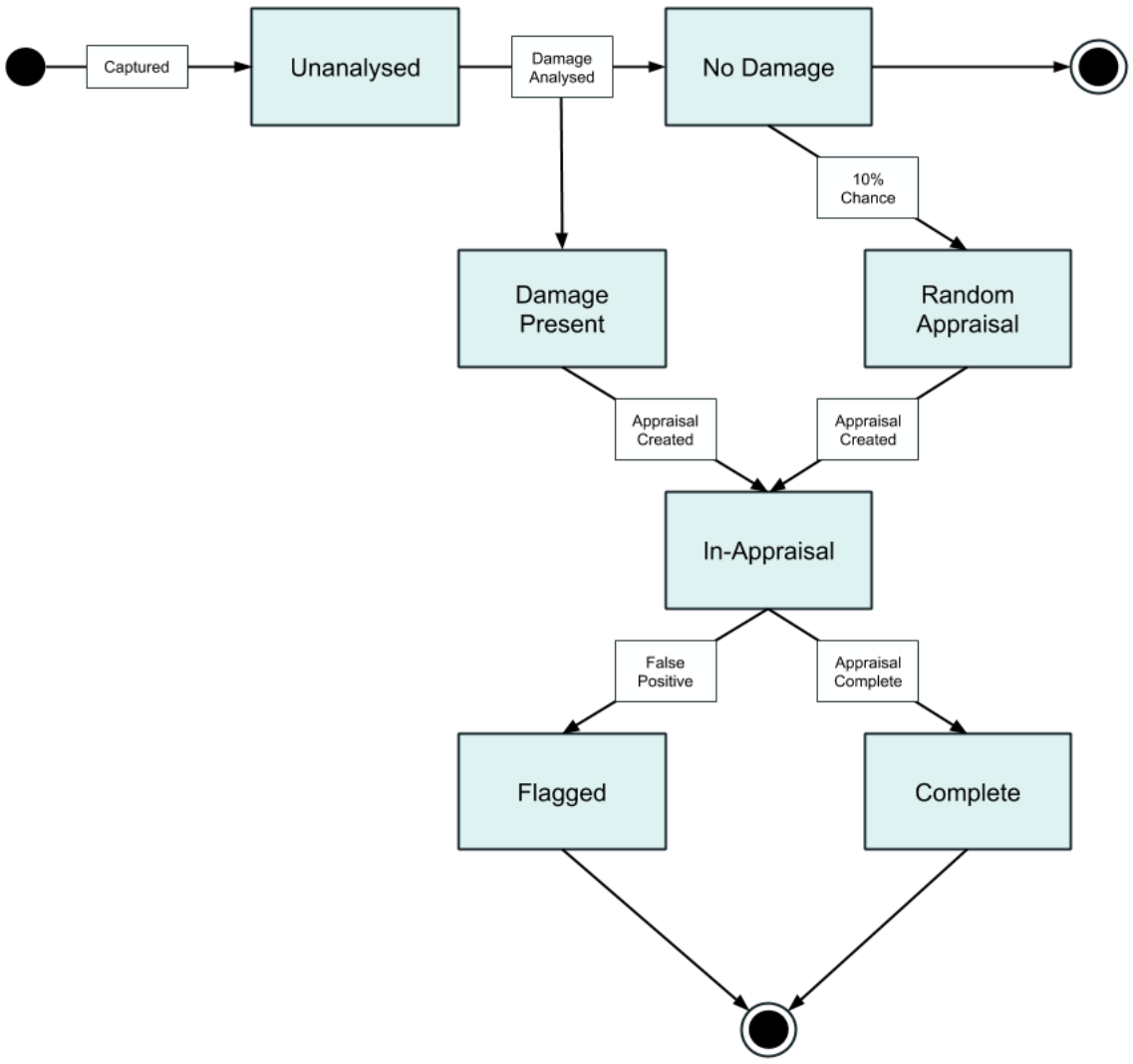


3.1.4 Data Flow Diagram



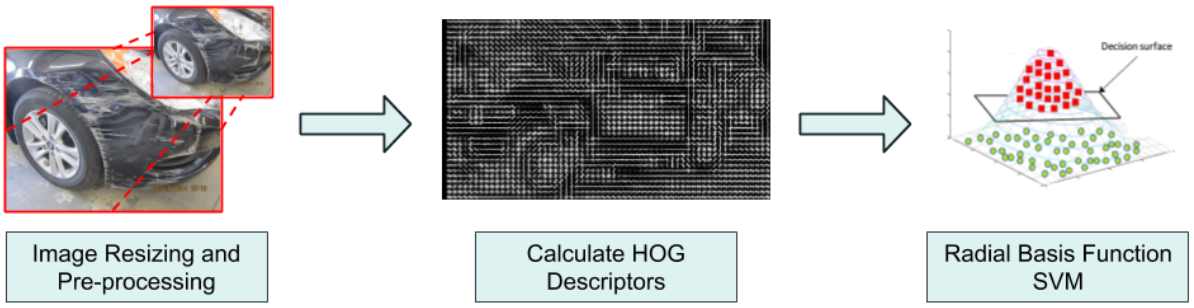
This data flow diagram provides a visualisation of the flow of data through the various components of the system. Beginning with the images being captured by the enclosure, and ending with an employee viewing or flagging an appraisal.

3.1.5 State Diagram



The state diagram describes the behavior of a system. The diagram displays the various states which can be held within the system. More specifically, this state diagram depicts the various states for an image which has been captured by the enclosure and has been transferred for analysis.

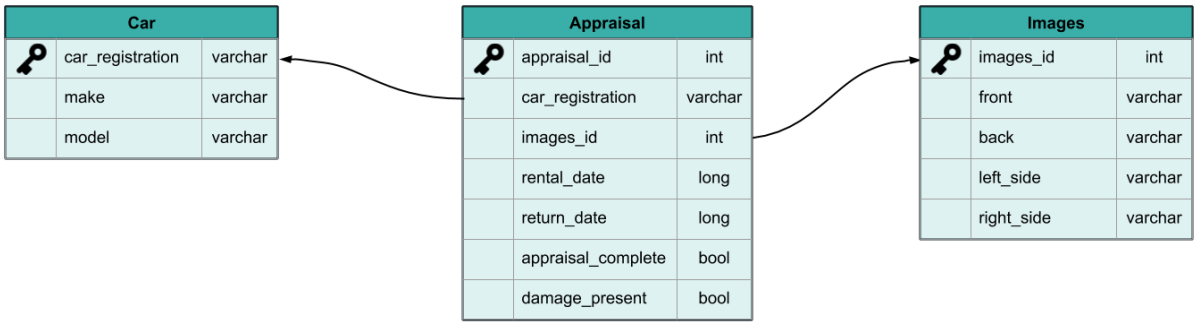
3.2 Damage Analysis Model Design



The use of HOG descriptors over SURF features allowed for the gradient lighting of dents to be analysed more accurately. Feature detectors generally highlight areas of interest which tend to be areas of high contrast between pixels. Dents can have very little contrast over a large area which means that feature detectors perform dent detection poorly.

Some of the previous models are outlined in the testing report submitted along with this technical manual.

3.3 Database Design

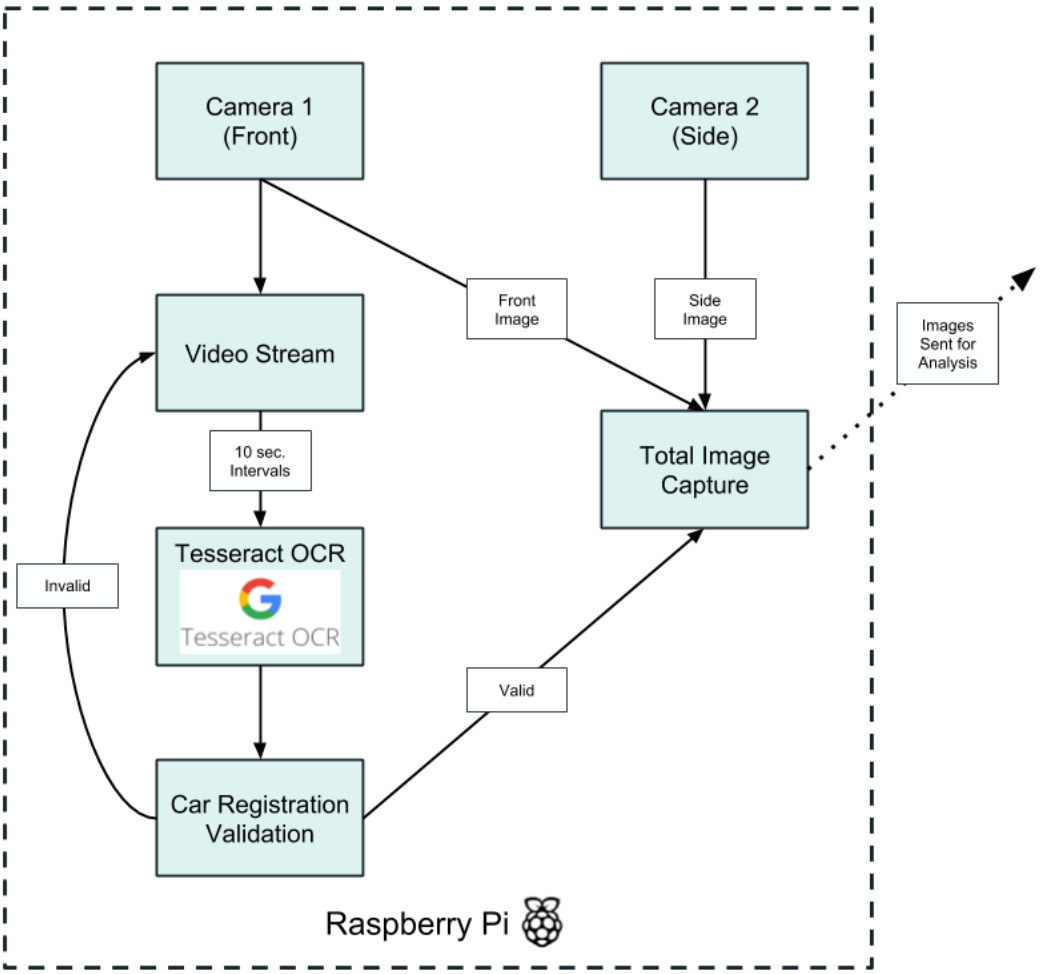


The main table of the system is the appraisal table, it stores the information related to each of the appraisals which must be evaluated by rental company employees. It contains foreign keys to each of the other two tables. In a production system, the rental_date and return_date would be populated with information from the rental companies rental databases, allowing for more in-depth knowledge of each rental.

The car table contains information related to each of the cars which are leased out in the rental companies. Information, such as car make and model, are present in this table. This information is displayed to employees when they are making an appraisal.

The images table contains file paths to each of the images relevant to an appraisal. It is best to store a reference to an image in the database rather than the image itself as storing them the image in the database can greatly increase the size of the database and increase its efficiency.

3.4 Enclosure Design



The enclosure is the part of the system which captures the images to be appraised. In order to do this, we must find out what car is being presented. The enclosure intakes a video stream from the camera facing the front of the vehicle, this video stream is parsed every 10 seconds using Google's Tesseract OCR engine (using the OCR engine allowed me to further focus on the implementation of the damage model without having to worry about completing an additional OCR project).

The strings which are parsed by the OCR engine are then validated to match the Irish reg plate of YYY-CC-SSSSSS. In addition to this, the system checks whether the reg plate matches a valid car reg in the database. If a valid registration plate is detected, images are captured from multiple sides of the car to be analysed.

4 Implementation / Technologies

The following sections will outline how various technologies were used to develop the functionality of the system. Sample code will be provided where applicable.

4.1 Python / OpenCV / Raspberry Pi

The damage analysis model was developed using the Python wrapper for OpenCV. I chose OpenCV as it is one of the most popular libraries for computer vision. More specifically, I chose the Python wrapper for OpenCV as Python allowed me to prototype new versions of damage analysis models with greater ease. OpenCV allowed me to use premium algorithms such as SURF and HOG without the necessity of implementing these from scratch.

I used the OpenCV machine learning modules to train the SVM using a Radial Basis Function Kernel for the final model. Some of the previous model iterations are outlined in the testing report submitted along with this project.

The code below prepares the images to be used for training the SVM. It iterates through each of the image files in the *dataset/train* directory and extracts the HOG descriptors for the image. These HOG descriptors are stored in a numpy array, the correct category for each image is stored in a separate numpy array.

```
for category, files in imagesByCategory.iteritems():
   for imageFile in files:

       print (category + ": " + str(imageCount), end='\r')
       sys.stdout.flush()

       filePath = "{}/{}/{}".format(TRAIN_IMAGES, category, imageFile)

       image = cv2.imread(filePath)

       image = cv2.resize(image, (128, 128))

       hist = hog.compute(image)

       trainData.append(hist)

       if categoriesAsNumbers[category] == 1 or categoriesAsNumbers[category] == 2:
           trainLabels.append(1)
       else:
           trainLabels.append(0)

       imageCount += 1
```

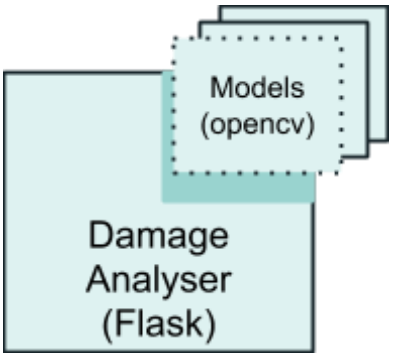
As this is a slow process to complete, the current file number is outputted to the terminal in order to gauge how far along in the process the the script is. The processed images and labels are then shuffled, followed by training of the SVM.

Python and OpenCV were also used in development of the Raspberry Pi enclosure. A video stream is analysed for a registration plate using Google's Tesseract OCR engine. When the plate is detected and validated, images are taken from various angles surrounding the car and sent to be analysed.

4.2 Python / Flask

The damage analysis model is deployed using Python and Flask. Deploying the model using flask has allowed me to continue prototyping new models with Python without having to convert the model to another language, such as Java, before deploying. This enabled me to move quickly while I was developing the project. The model can then be used by interacting with the flask endpoints through HTTP requests.

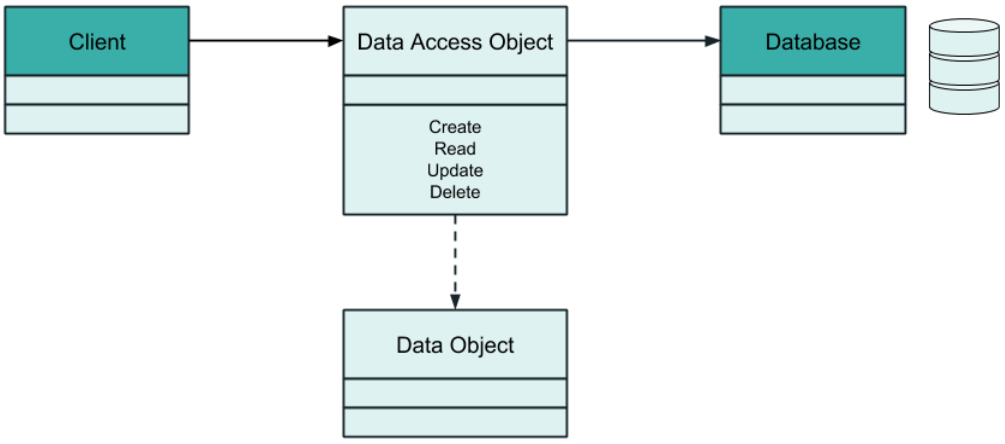
Flask allowed me to slot in improved versions of the damage analysis model with ease, as described in this section of the system architecture.



4.3 Java / Spring

The main REST API for the system is written in Java using the Spring framework. The main reason for using the spring framework came in Spring's Data library using the Java persistence API, and the structured manner in which it allows you to interact with your database. Spring forces you to create data access objects which will hold the information stored in your database. Below is a small class diagram of the Car class which interacts with the car table of the MySQL database.

Data Access Object Pattern



The Spring REST API is used to supply the data to populate the pages of the web application with past and present appraisals for cars. The API offers endpoints which allow the web application to make updates to the database. Spring is also an industry standard tool used in the development of REST APIs. Creating the main REST API using Spring allowed me to gain some valuable experience in how Spring works.

The below code snippet is just one of the many GET endpoints developed for the REST API. This endpoint is responsible for returning the content used to populate the damage report page.

```
@RequestMapping(value="/getReport", method = RequestMethod.GET)
public AppraisalAccess getDamageReport(@RequestParam String stringAppraisalId) {

   int appraisalId = Integer.parseInt(stringAppraisalId);

   AppraisalAccess report = appraisalRepository.findByAppraisalId(appraisalId);

   return report;

}
```

As you can see, this code snippet makes a call to the Appraisal Repository, which is the Data Access Object for the appraisal table of the database. This call, in turn, performs a query to the database which returns all information relevant to the appraisal with the matching appraisal ID.

The following code snippet is used to mark an appraisal as complete.

```
@RequestMapping(value = "/markAppraisalComplete", method = RequestMethod.PUT)
public ResponseEntity<?> markAppraisalComplete(@RequestBody String stringAppraisalId) {
   int appraisalId = Integer.parseInt(stringAppraisalId);

   AppraisalAccess appraisalAccess = appraisalRepository.findByAppraisalId(appraisalId);

   appraisalAccess.setAppraisalComplete(true);

   appraisalRepository.save(appraisalAccess);

   return ResponseEntity.accepted().build();
}
```

The code gets the relevant appraisal from the database using the appraisals unique ID. Then, using the returned appraisal data access object, it sets the relevant appraisals appraisal complete status to true. The updated data access object is then saved to the database.

While developing the Spring API, I made use of *Postman*, which allowed me to perform AdHoc testing of the API as certain functionalities were completed.

4.4 MySQL Database

For the database of the system, I created a MySQL database which was used to house the relevant information for each of the appraisals and the associated car's information. The formation of relational databases lends itself well to the way in which I wanted to structure the data. Rather than storing images directly in the database, I instead stored a reference to the image in the images table of the database and stored the actual image in local storage of the Spring server. The database design is discussed previously in this report.

4.4 TypeScript / Angular Web Application

Angular 7 was used to develop the front-end web application for the system. As stated previously, the web application receives content from the Spring REST API. It is then parsed using typescript and used to populate pages of the application such as the damage report page. Pages in the web application are dynamically populated when data is received.

```
<div class="awaiting-appraisal">
 <a [routerLink]="['/report',car.appraisalId]" *ngFor="let car of awaitingAppraisals">
   <div class="appraisal">
     <p class="appraisal-heading">Appraisal No. {{car.appraisalId}}</p>
     <img src="http://localhost:8080/getImage?filepath={{car.imagesAccess.front}}">
     <p id="reg-plate">{{car.carAccess.carRegistration}} ({{car.carAccess.make}},
{{car.carAccess.model}})</p>
     <p>{{car.rentalDate}}  -  {{car.returnDate}}</p>
   </div>
 </a>
</div>
<div *ngIf="noAppraisalsFound" class="error">
 <h2>No Appraisals Found</h2>
</div>
```

The above Angular HTML template is for the appraisal page, which contains the lost of all appraisals to complete. It makes use of angular for loops to populate the page with data. The information from the database are accessed using {{variable_name}}. The content of the page is hidden if no appraisals are found.

With the Angular front-end, I strived to build an intuitive and usable interface which could be picked up by rental company employees with ease. Throughout the application, I aimed to keep the layout and design consistent and familiar to the user.

4.5 Google Cloud Platform

Using the available free Google Cloud Platform credits, I deployed the training section of the damage analysis model to the cloud. The initial $300 credits as well as the additional €50 allowed me to use some powerful hardware while training. Training on the cloud allowed me to shorten the amount of time necessary for me to train each iteration of the model, while also freeing up my personal laptop to allow me to develop additional areas of the project while training still took place.

5 Problems and Resolutions

5.1 OpenCV Raspberry Pi

Problem: The most recent version of OpenCV(4.0) does not fully support the Raspberry Pi, and as such, OpenCV cannot be installed easily with the use of pip for instance.

Solution: The necessary version of OpenCV - and its Python wrapper - must be built on the Raspberry Pi itself. The repository for OpenCV must be forked or downloaded and the make commands must be run. This is a time consuming process due to the power available on the Raspberry Pi's CPU and can take upwards of 4 hours.

5.2 Lack of Available Data

Problem: There is a lack of readily available datasets which can be used for the creation of a damage detection and analysis machine learning model.

Solution: It was necessary to create a new dataset myself. To do this, I made use of the Stanford Car Dataset, which contains thousands of images of cars. I then had to manually categorise these images as either undamaged or damaged. As most of the cars in this dataset were undamaged, further effort was required to increase the amount of damaged images contained in the dataset. I used web-scraping tools to harvest data from Google Images which could be used in the dataset.

The quality of this dataset is discussed further in the testing report submitted alongside this technical manual.

5.3 Deployment of the model

Problem: The initial system was developed using a Spring REST API. It would not be feasible to deploy the damage analysis model using Java and it would not be feasible to convert the model to OpenCV Java as it is much less supported than OpenCV Python.

Solution: I used Flask and Python to deploy the model. This allowed me to focus on improving the various iterations of the model by deploying and prototyping the model using the same language. The use of these programs lowered the amount of time necessary to develop iterations of the model.

5.4 Out of Memory Error While Training

Problem: While training the model on my personal laptop, I encountered a number out of memory errors. This caused the training to fail and any progress made in the training to be deleted.

Solution: In order to solve this issues, I performed a number of changes. The code was altered to delete any variables which would not be used after training in order to clear some extra memory space. In addition, I moved the training of the model to train on the Google Cloud Platform using the free credits we received. Use of GCP allowed me to use more powerful hardware for training, while also freeing my personal laptop to allow me to complete additional aspects of the project.

5.5 Storing a Large Number of Images

Problem: The overall system would generate a huge amount of images over its lifespan. For each appraisal, a number of images are captured and these images need to be saved. Storing these images in the database itself would cause queries to execute at a much slower speed.

Solution: Instead of storing images in the database itself, references to the images are stored in the database. Each image itself is stored in the local storage of the Spring server itself. The path to the image is stored in the database, images are saved of the front, back, left, and right side of the car. This method of saving images allows them to be accessed much quicker than when saved directly in the database.

5.6 SURF Keypoint Detection for Dents

Problem: SURF keypoints are generated using the amount of contrast between smaller areas of pixels. This works well for damage to cars such as scrapes, where there is a higher level of contrast between the paint of the car and the scrape itself. However, it is not as sufficient for dents to the body work of a car as dents are generally made up of a gradient of similar colour rather than a sharp contrast of colours.

Solution: HOG descriptors generate features globally rather than locally. They aim to understand the flow of the pixels rather than the difference of individual pixels to one another. Dents can be viewed as a gradient change of colour on the body of a car. For this reason, HOG descriptors are much better suited for the detection of both dents and scrapes.

6. Installation Guide

As a prerequisite to the following deployment sections you must fork or download the git repository, in order to access the necessary source code.

6.1 Web Application

Change directory into *web-application*

The web application makes use of npm in order manage dependencies. The necessary dependencies can be installed using *npm install.*

The command *ng serve* will serve the front end on the localhost of your machine. The web application will be accessible on *http://localhost:4200/*

NOTE: The content will be populated from the rest api.

6.2 Spring REST API

Change directory into *restapi*

The rest api uses maven to handle dependencies. The necessary dependencies can be installed using *mvn install.*

The rest api can be compiled using the *mvn clean build* command.

The rest api can be deployed to localhost using the following command:

sudo java -jar target/restapi-1.0-SNAPSHOT.jar

6.3 Model Deployment

Prerequisites: Python OpenCV 4.0 and Flask are required to be installed.

Change directory into *model-deployment.*

The following command will display the model to localhost:

sudo python app.py

NOTE: The flask endpoint will be made accessible by any computer on the same network.

6.4 Enclosure

Prerequisites: Python OpenCV 4.0 must be built from source on the Raspberry Pi. Tesseract OCR must be installed on the Raspberry Pi

Download the code from the *enclosure* directory onto a Raspberry Pi. Change directory into *enclosure* on the Raspberry Pi.

Run the following command to perform video parsing on the Raspberry Pi:

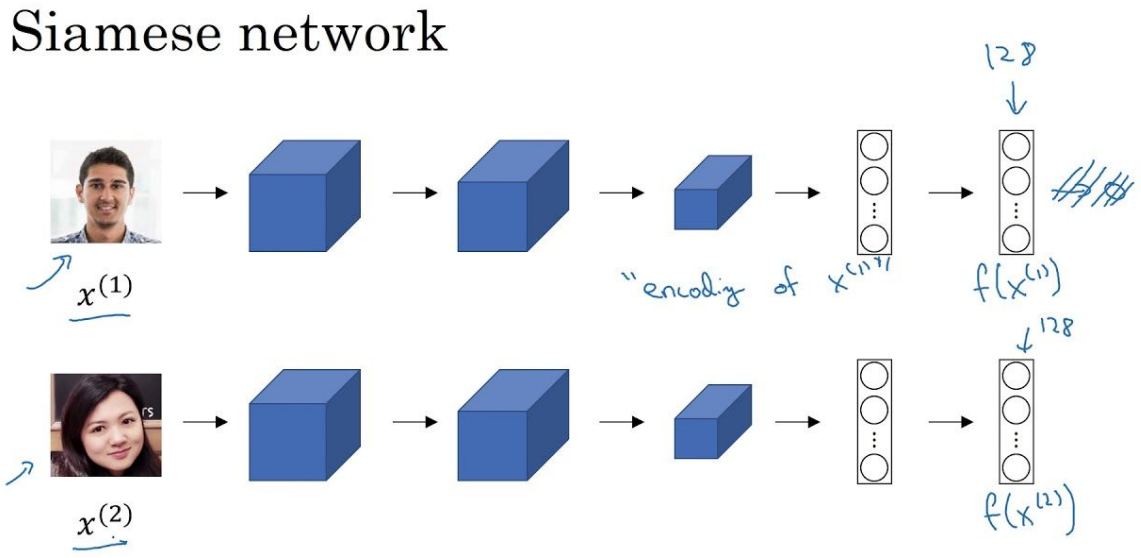
sudo python videoInput.py

7. Future Work

This project has significant potential for future work, both in improving its current feature list and adding new features to the system.

The damage analysis model could be improved with the use of deep convolutional networks in conjunction with the gathering of more data. Convolutional neural networks have been shown to perform better than more traditional machine learning algorithms when provided with enough data. This could lead to a much more accurate system.

Moreover, these convolutional neural networks could be used to devise a system which - rather than classifying images of damaged or undamaged - would detect exact areas of difference in before and after photos of cars on a rental. This could be achieved with the use of siamese neural networks, which have been used to produce very good results in facial recognition problems. In order to achieve this, a new dataset would need to be constructed which contains before and after images of damaged cars.



Working cooperatively with car rentals companies would enable this system to integrate with their current appraisal systems to complete full damage appraisals through the application itself. A working arrangement with these companies would also provide us with more data through their current archive of damaged cars, which could then be used to further improve the model's accuracy.

Further in the future, it would be possible to develop a model which is capable of not only detecting damages to a car but also performing appraisals in order to gauge a cost, which could be applied to the customer's rental.