Car Damage Analysis

Testing Report - David Weir (Student No. 15432642)

Testing Report

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 Validation

The goal of this project was to streamline the process by which cars are returned in the car rental industry. Through conversations with employees of two of these car rental companies, I have discovered that the current process takes roughly 5 minutes from start to finish. They did say, however, that this can vary depending on the individual employee and the company's policies in relation to 'fair wear and tear'.

To validate this project, I timed the entire process of a car entering the enclosure and the enclosure receiving a prediction. The total time for the system to analyse a car for damages was a maximum of 2 minutes, depending on performance of the registration plate detection running on the Raspberry Pi**.** This decrease in time taken provides clear validation for the development of this project.

NOTE: The timing of this system could be improved with the addition of hardware which is more powerful than a Raspberry Pi performing registration plate detection.

In addition to the decrease in time it takes to analyse a car, this system provides an additional layer of standardisation to the process. With the introduction of a computerised system comes a decrease in the amount of variation in what constitutes damage or 'fair wear and tear' to a car, as the analysis becomes less subjective to the person performing it.

2 Use Case Testing

Use Case No.	Title	Steps	Result
1	Damage Appraisal	1. Employee enters web application. 2. Appraisal page is displayed, containing a list of cars to be appraised. 3. Employee clicks on an appraisal. 4. Damage report page is displayed, containing images of the car. 5. Employee uses the images provided by the application during the appraisal. **Expected Result:** Car is marked as appraised.	PASS
2	Appraisal Dispute	**Precondition:** Damage has been detected on the car. Customer disputes damage. 1. Employee enters web application. 2. Enters the search page. 3. Enters a valid Irish car registration number. 4. Search results are displayed. 5. Employee clicks on the valid search result. 6. Timeline is displayed, showing the past appraisals for a car. 7. Employee chooses the appraisal in question, and the damage report page is displayed. **Expected Result:** A complete damage report page is displayed, containing images of the car.	PASS
3	Report Flagging	**Precondition:** Damage analyser has incorrectly reported a car as damaged 1. Employee enters web application. 2. Appraisal page is displayed, containing a list of cars to be appraised. 3. Employee clicks on an appraisal. 4. Damage report page is displayed, containing images of the car. 5. Images do not contain damage to the	PASS

		car. 6. Employee flags the appraisal as incorrect. **Expected Result:** Appraisal is flagged in the database, the appraisal is then used in order to retrain the model.	
4	Returning a Car	1. Customer car enters the enclosure. 2. Enclosure captures the registration plate of the car. 3. Enclosure captures images of the car from various angles. 4. Images are sent to be analysed. 5. Damage analyser completes the analysis. **Expected Result:** Customer car is analysed successfully as containing damage. Appraisal created for the web application.	PASS
5	Adding a New Car	1. Employee enters web application. 2. Enters add car page. 3. Employee fills the form with relevant car information. 4. Employee clicks submit. **Expected Result:** New car is added to the database to be used when creating appraisals in the future.	PASS

3 Unit / Integration Testing

Due to the number of technologies used during development of this project, each aspect of the project had to be tested using the most suitable framework for the development technology.

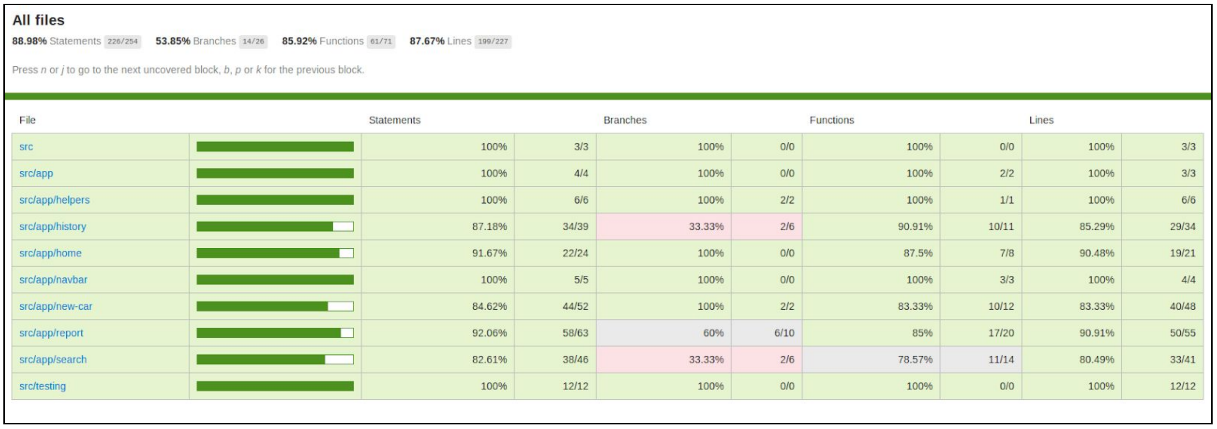
The tests cover the functionality associated with the web application and overall system. They do not include the code which trains and tests the damage detection model. Where applicable, these tests explore both the happy and sad paths of the functionality in order to broaden the scope of testing.

3.1 Angular / Web Application

Angular offers a built in module which allows for testing using the Jasmine test framework and run using Karma, a JavaScript test runner. These tests verify that functions perform correctly and where necessary, update the user interface with the appropriate data.

In total, there are 35 tests written using the angular testing module, which achieve an overall coverage of 88.98 for the web application.

These tests can be run from the *'web-application'* directory using the *'ng test'* command. The coverage report will be located in the generated directory named *'coverage'.*



All files

88.98% Statements 226/254 **53.85%** Branches 14/26 **85.92%** Functions 61/71 **87.67%** Lines 199/227

Press *n* or *j* to go to the next uncovered block, *b*, *p* or *k* for the previous block.

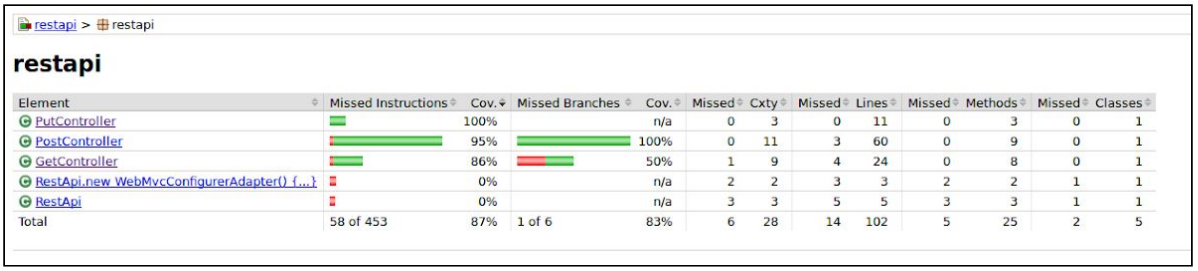
File	Statements		Branches		Functions		Lines	
src	100%	3/3	100%	0/0	100%	0/0	100%	3/3
src/app	100%	4/4	100%	0/0	100%	2/2	100%	3/3
src/app/helpers	100%	6/6	100%	2/2	100%	1/1	100%	6/6
src/app/history	87.18%	34/39	33.33%	2/6	90.91%	10/11	85.29%	29/34
src/app/home	91.67%	22/24	100%	0/0	87.5%	7/8	90.48%	19/21
src/app/navbar	100%	5/5	100%	0/0	100%	3/3	100%	4/4
src/app/new-car	84.62%	44/52	100%	2/2	83.33%	10/12	83.33%	40/48
src/app/report	92.06%	58/63	60%	6/10	85%	17/20	90.91%	50/55
src/app/search	82.61%	38/46	33.33%	2/6	78.57%	11/14	80.49%	33/41
src/testing	100%	12/12	100%	0/0	100%	0/0	100%	12/12

3.2 Java / Rest API

In order to test the Java Rest API, I made use of JUnit in addition to the testing framework for Spring, and Mockito to mock http calls. The JUnit tests cover the functionality of the rest api such as saving images to the local storage of the machine an interactions with the database.

Using 14 individual test cases, the JUnit tests achieved an overall coverage of 87% for the Rest API. Some of the lines which have not been covered by these tests include setup for the Spring server, and a function which enables Cross-origins requests from the web-application and deployed damage analysis model.

These tests can be run from the *'restapi'* directory using the *'mvn test'* command. The coverage report will be located in the generated *'target'* directory under a directory named *'site'.*



restapi > restapi

restapi

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
PutController		100%		n/a	0	3	0	11	0	3	0	1
PostController		95%		100%	0	11	3	60	0	9	0	1
GetController		86%		50%	1	9	4	24	0	8	0	1
RestApi.new WebMvcConfigurerAdapter() {...}		0%		n/a	2	2	3	3	2	2	1	1
RestApi		0%		n/a	3	3	5	5	3	3	1	1
Total	58 of 453	87%	1 of 6	83%	6	28	14	102	5	25	2	5

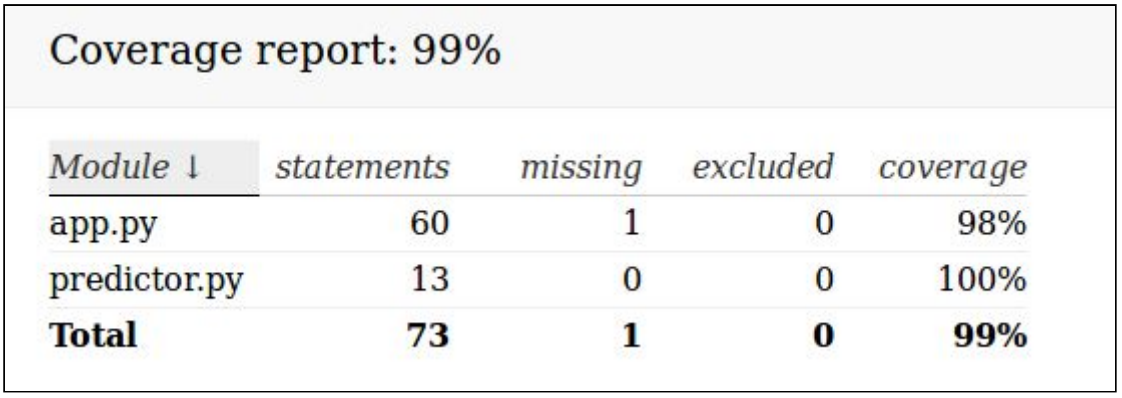
3.3 Python / Model Deployment

As the damage analysis model had to be deployed using Python, pytest was used as the testing framework for managing testing of the model deployment. In addition, the Python mock library was used in order to mock some http calls for testing purposes.

With 14 individual tests, the pytest tests achieved an overall coverage of 99% for the model deployment.

These tests can be run from the *'model-deployment'* directory with the command *'pytest'.* Coverage can be calculated using the python coverage library with the following commands:

1. *coverage run --source=. --omit=./test_app.py -m pytest*
2. *coverage html*
3. The report will be available in the generated *'coverage'* directory.



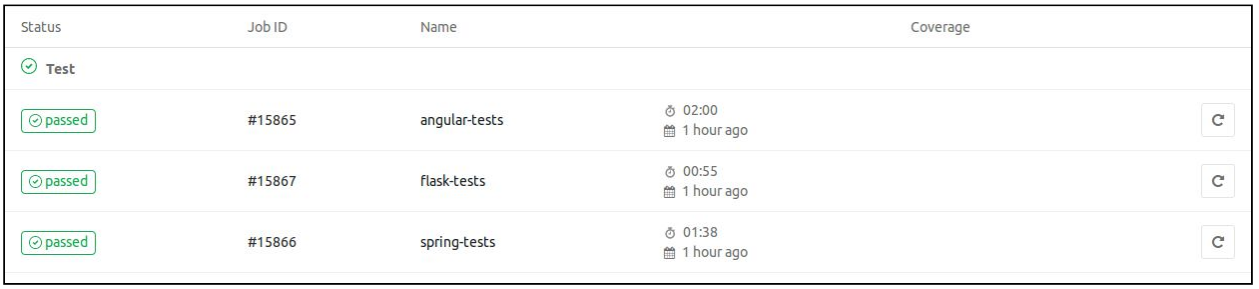
Coverage report: 99%

Module ↓	statements	missing	excluded	coverage
app.py	60	1	0	98%
predictor.py	13	0	0	100%
Total	**73**	**1**	**0**	**99%**

3.4 Pipeline / Gitlab CI

In order to minimise the number of bugs introduced during development, each of the above tests was run as a job in the GitLab CI pipeline. Each of the jobs: angular-tests, flask-tests, and spring-tests pulls a different docker image which contains the dependencies necessary for it to execute the tests.

The introduction of the pipeline allowed me to ensure that all tests would be run whenever code was committed. I would not merge a branch into master unless the most recent pipeline was a success. Below is a screenshot of a completed pipeline run.



4 Model Evaluation

A substantial portion of the time for this project was dedicated to the research and development of the damage analysis model, which predicts whether an image of a car contains damage or not. The following sections will explore how the model was evaluated during its development, possible issues with the current model, and how the model changed during the course of the project.

4.1 Accuracy Testing

The damage analysis model could be tested with the use of a validation dataset, which contained a number of images of each category. These images were never used during training of the model, and thus, were new to the model. Using a validation dataset decreases the possibility of the model becoming overfitted to the data which is provided to it.

The final validation portion of the dataset contained a total of 580 images, 280 damaged and 300 undamaged. The damaged section contained 100 dented and 180 scraped images.

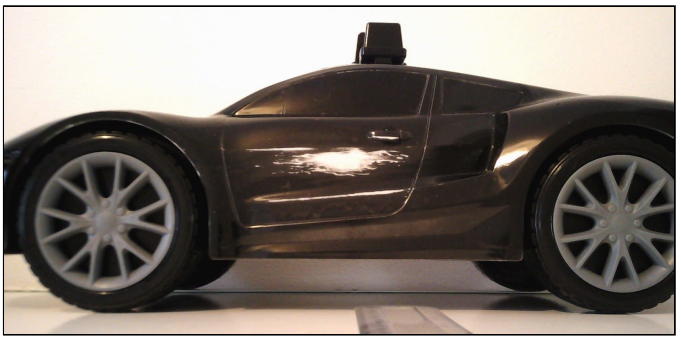
In the end, the damage analysis model achieved an accuracy of roughly 80% when accuracy is measured against the validation portion of the dataset created for this project.

4.2 Batch Real-World Model Testing

In order to get an idea as to how the model would perform in a real world setting, a script was developed to be used in conjunction with the hardware enclosure. This script captured images for prediction of a model car in the enclosure every 10 seconds and outputted the prediction result to the terminal.

Automating this process allowed me to alter the environment of the enclosure by changing the lighting setup around the hardware. The lighting was changed using a spotlight to vary the amount of glare which was on the paint of the model car, and a sheet to dim light sources to simulate night, dusk and dawn.

The model car, with considerable damage, can be viewed here:



Although the model car contained noticeable damage, the damage analysis model struggled to recognise any damage under certain lighting conditions. Some of the possible reasons for this will be discussed in the following section.

4.3 Analysis of Data

Some of the difficulties with this project lay in the data necessary for creation of the damage analysis model. While researching this project, I found that there are very few - if any - available datasets for damaged cars. For this reason, I opted early on to create a dataset which would be appropriate for my needs.

As the foundation for my dataset, I used the Stanford car dataset, available at https://ai.stanford.edu/~jkrause/cars/car_dataset.html. This dataset was then cleaned in order to remove any unsuitable images, and the images were categorised into damaged or undamaged. The majority of the cars from this dataset fell into the undamaged category. In order to collect additional images of damage to cars, I web-scraped images from Google and categorised these images into scraped and dented.

There is an inherent issue with the dataset I created however, that is whenever someone is uploading an image of their damaged car to the internet, they tend to only include images of the immediate area surrounding the damage. This becomes an issue when these images of damage are compared to the undamaged images of the dataset, which as stated previously come from the Stanford car dataset.

The vast majority of damaged cars contained zoomed / cropped images of the local area surrounding the damage, whereas the undamaged images tend to contain the entire car.

The example below illustrates the differences between categories in the dataset.

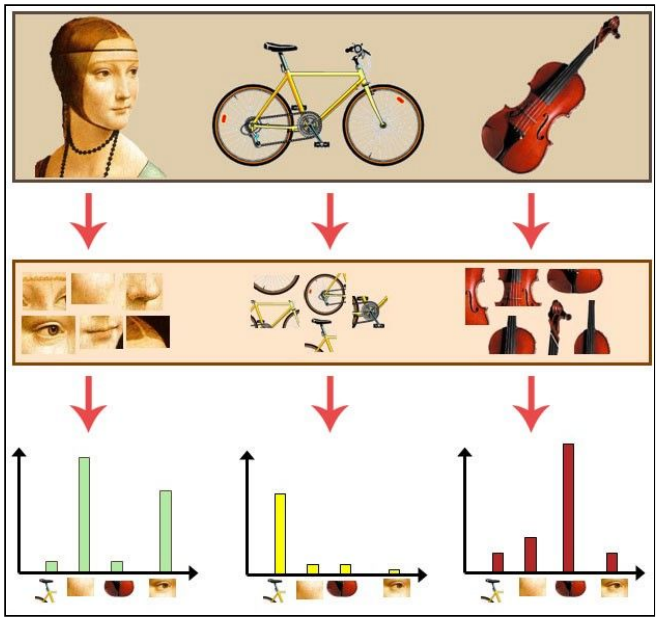
While measures were taken to combat differences in the data while training - such as zooming out the damaged images and cropping the undamaged images - it is difficult to say whether they greatly improved the real-world performance of the system.

4.4 Model Iterations

The development of the damage analysis model was an iterative process, I did not arrive at the final model straight away.

Initially, I planned to perform a comparison of the SURF keypoints in order to detect damage in the images. The SURF feature matching algorithms of OpenCV did not perform well with my dataset. In the example below, you can see that a number of the SURF key points have matched to incorrect points in the images. In order to compare these key points, I attempted to implement a distance metric which could calculate a similarity score for images of damage. The results of this were quite poor which led to me exploring other avenues.

Next, I began researching the possibility of using classification algorithms to detect whether an image contained damage or not. The first method is called bag of visual words, a computer vision version of the bag of words information retrieval algorithm. This method again made use of SURF key points, where it attempted to calculate the common key points associated with each of the categories using K-means clustering. The image below provides a visual representation of how the bag of visual words algorithm works.

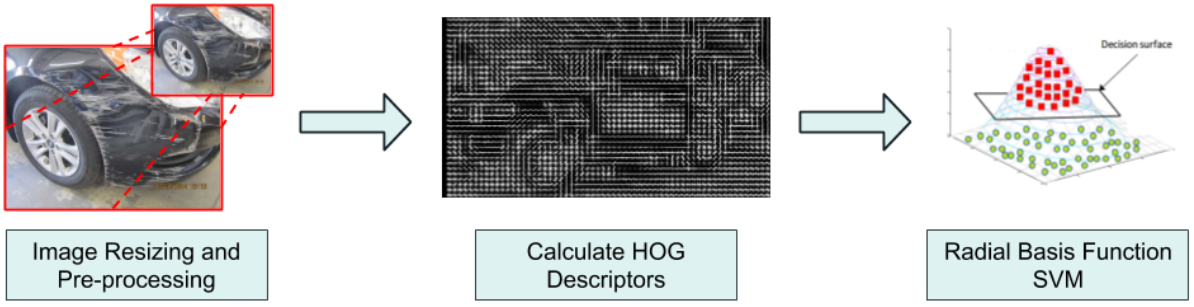


While the bag of visual words implementation did provide decent accuracy when tested against the dataset I created, it still struggled in some areas. The model achieved an accuracy of 60%. When delving into the predictions of the dataset, it became clear that the model was failing to detect any images which contained only dents. This is due to the fact that SURF key points rely heavily on the contrast between pixels to generate a keypoint. While damage such as scrapes have a high amount of contrast, dents have

very little contrast and have a much softer gradient between pixels. In order for the model to be able to detect dents at a higher rate, a new form of key point generator would need to be implemented in the project.

The final model attempt implemented for this project involved the use of Histogram of Oriented Gradient descriptors, used to classify images as either damaged or not damaged. The HOG descriptors perform much better at damage detection as they calculate regions of interest at a more global scale, rather than the local scale of SURF key points.

HOG descriptors are harvested for each image and these images are used to train a Support Vector Machine, using a Radial Basis Function kernel. As stated previously, this method achieved an accuracy of 80% after some tuning of the SVM parameters.

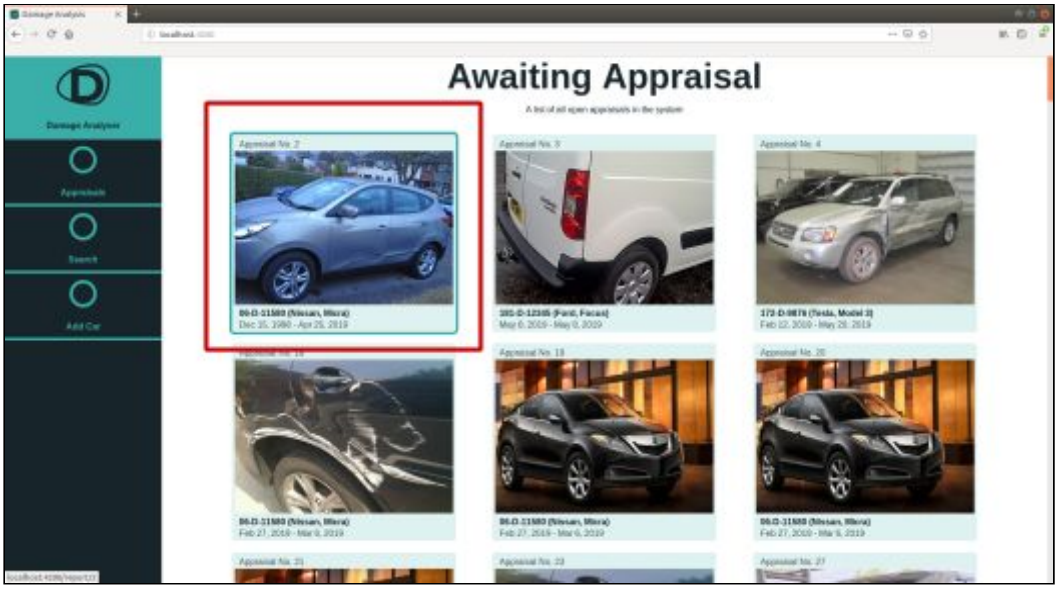


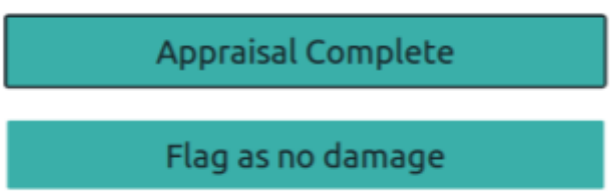
Even still, the final is not perfect as it sometimes can predict an image containing damage as undamaged and vice versa. To combat this, a service was added to the model deployment section of the system which sends 10% of all cars to be analysed by a human employee. These can then be used to further train and improve the model, adding images it has incorrectly identified to the dataset.

5 Heuristic Testing

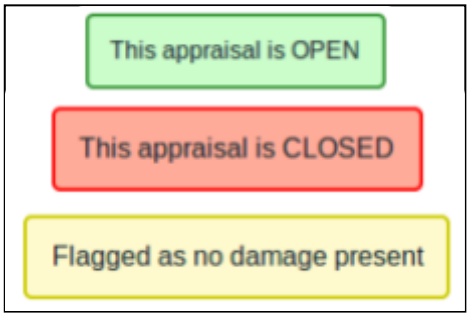
Heuristics provide a good measure for the usability of an application. These heuristic tests can be performed without the need for real world users. Instead, they can be performed by the developers of an application to gauge its effectiveness. For this section, I used Nielsen's heuristics as described by Dr. Donal Fitzpatrick (https://www.computing.dcu.ie/~dfitzpat/nielsens-heuristics).

1. **Simple & Natural Dialog:** Dialog in the application is designed to be kept simple in order to ensure that users of all abilities can interact with the application. Descriptions are provided for each page in order to simplify the UI for employees.
2. **Speak the User's Language:** The technologies of the application are abstracted from the user, in order to provide a more familiar experience. For instance, the user is not told how damage on a car is detected. Instead, they are told that damage is present.
3. **Minimise the User's Memory Load:** The application functions in a very similar manner to the vast majority of sites on the internet. Users can maneuver through the application through the navigation bar along the side of the screen. As with many web applications, the user can return to the home page by clicking on the application logo in the top right of the screen.
4. **Consistency:** The application strives for consistency throughout. Clickable sections of the UI are denoted using a pale blue background which reacts to the users cursor with a darker blue outline. All buttons in the applications are styled in the same manner to ensure the user will know right away what their function is.

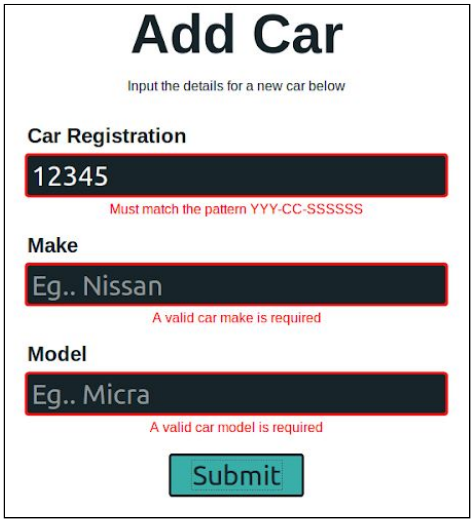




5. **Feedback:** Each aspect of the site provides feedback to the user when they interact with it. Clickable sections become outlined, a search results box is displayed when the users has successfully completed a search, and colour-coded flags are displayed to a user when they make changes to a damage report for a car.



6. **Clearly Marked Exits:** As the application is browser-based, users will be able to navigate to previous pages using the back arrow keys of the browser. As stated previously, the user can return to the homepage of the application by clicking on the logo of the application, which is a common navigation method in modern web applications.
7. **Good Error Messages:** Error messages are displayed where applicable throughout the UI. Users are prevented from entering invalid registration plates when adding a new car. An error message is displayed showing the user the correct format which must be followed for the registration plate.



8. **Prevent Errors:** When completing certain actions through the application - such as completing or flagging an appraisal - the user is asked to confirm their action. This is in place to reduce the likelihood of a user accidentally flagging an appraisal.

9. **Help and Documentation:** Users have access to the user manual for the application, which will walk them through how to complete certain tasks through the UI.

6 UI / Accessibility

6.1 Sight Impairments

The text sizes throughout the application are kept large to ensure that any users who have difficulties with their vision can read the information with ease. The gallery function on the report page also allows users to view larger versions of the images captured through the enclosure.

The colour scheme of the application was chosen in order for it to be both aesthetically pleasing and easy to use for all possible users. The colour pairs used in the web application provide a high contrast to ensure that users can view content easily.

The accessibility of the chosen colours was validated using an online accessibility checker, with the results shown below.

Note: WCAG refers to the Web Content Accessibility Guidelines for colour accessibility.

6.2 Motor Skill Impairments

User interactions are kept simple in the web applications to ensure ease of use for all users. All input fields and buttons are kept large to ensure users with motor skills impairments can interact with the application with ease. Feedback is provided by the buttons to inform users they are hovering over an interaction field.

Application features, such as the gallery, offer users two forms of interaction. Large on-screen buttons are present to change image and close the gallery, in addition to allowing users to use the arrow keys to change image and escape key to close the gallery.

7 User Testing

7.1 Plan

The users who participated in the user testing for this application were provided with a list of tasks which they were expected to complete. The users were timed as they completed these tasks so I could verify that the system was easy to understand.

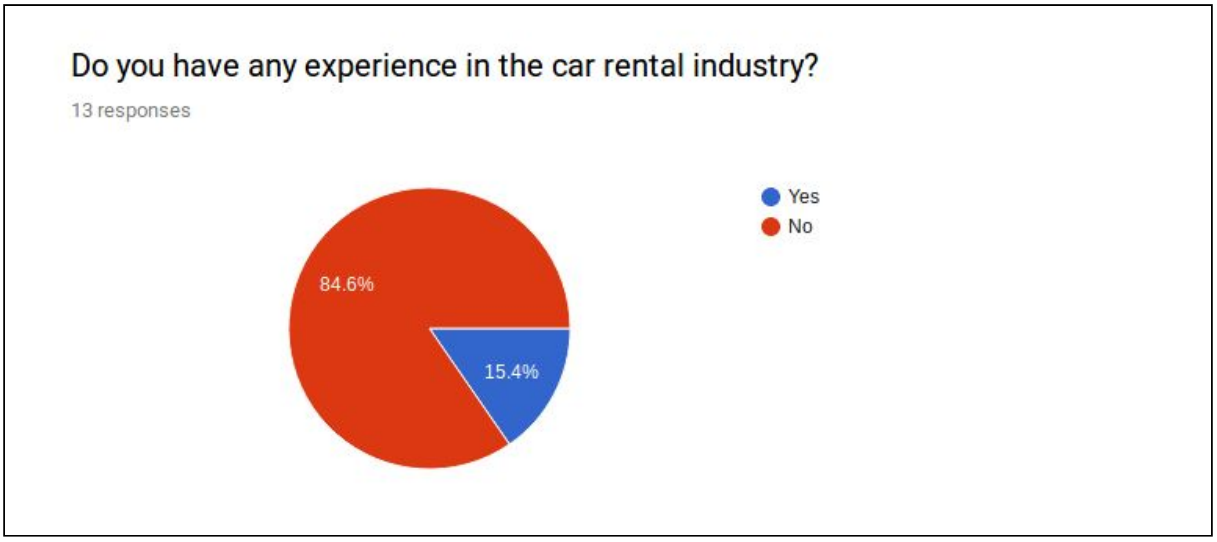
The tasks that the users were asked to complete cover the main functionality of the system, and follow the flow of the car return process used in industry. The tasks are below:

1. Return a car using the hardware enclosure.
2. View the newly created damage appraisal for the car.
3. Examine the images and complete the appraisal.
4. Add a new car.
5. Search for a car.

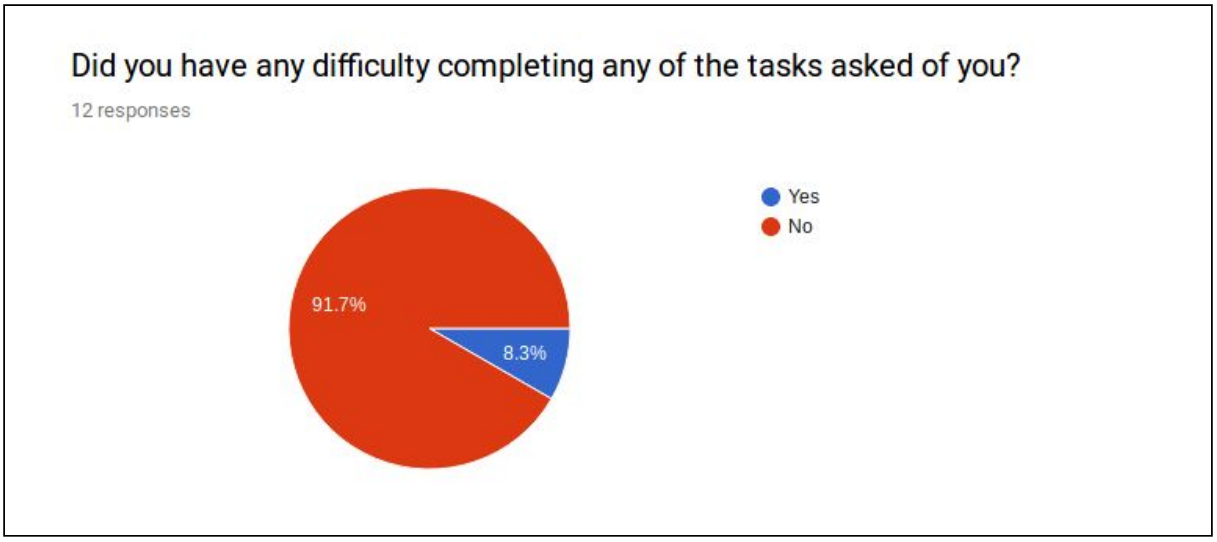
7.2 Outcome

In general, the users completed all of these tasks in 9 minutes. This is an acceptable time-frame for completing the tasks and it shows that the application is easy to become accustomed to. Access to the user manual allowed the test users to figure out how to complete the more ambiguous tasks with greater ease. A large portion of this time was dedicated to the first task due to the car registration detection running somewhat slowly on the Raspberry Pi hardware.

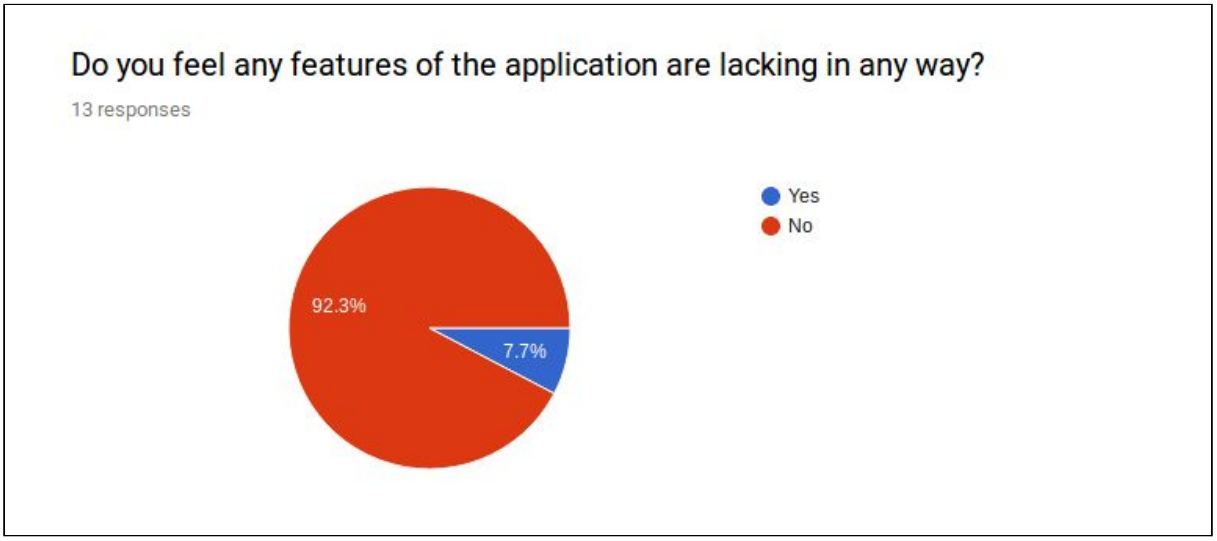
The majority of test users did not have experience in the car rental industry. Those who did have experience in the profession provided me with valuable insight for improvements that could be made to the system.



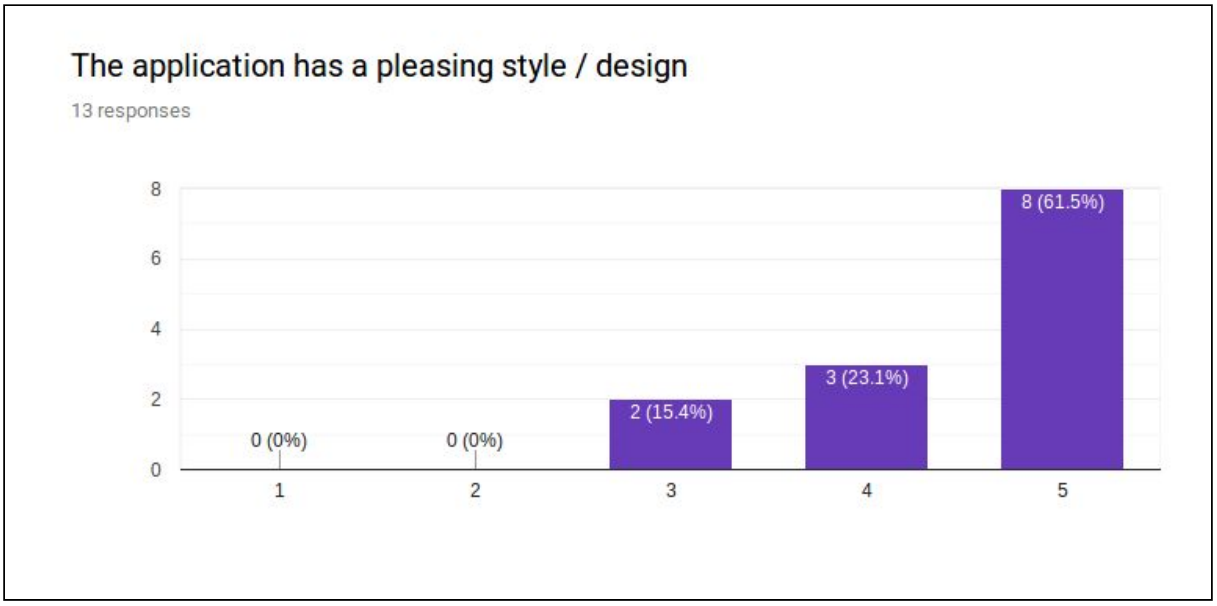
The majority of users did not struggle when completing the tasks assigned to them.



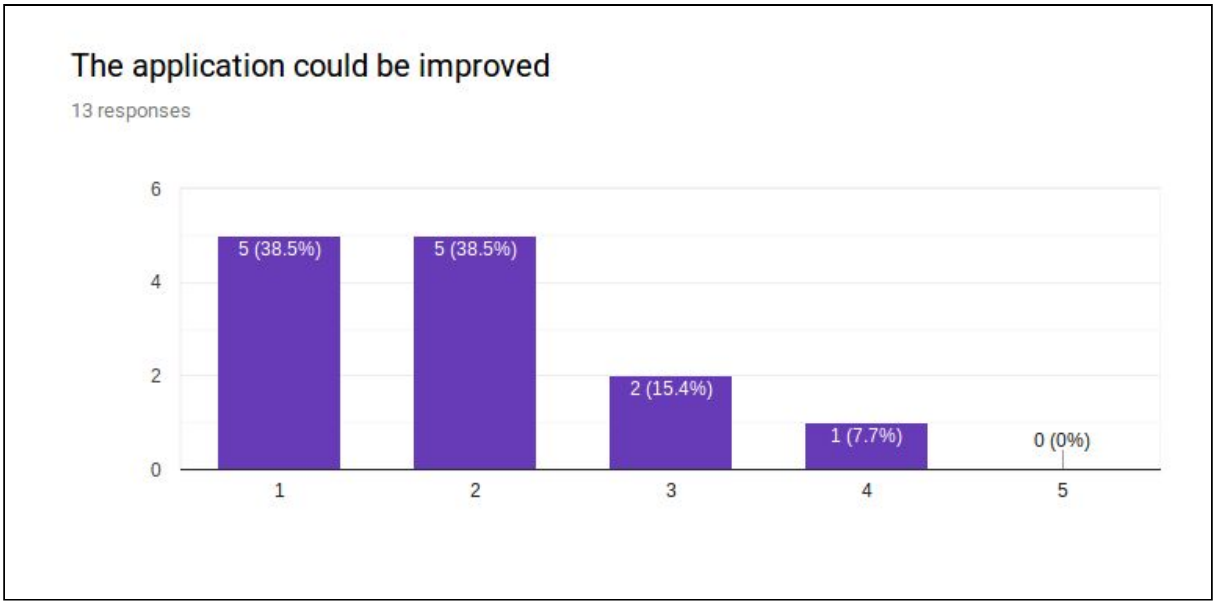
Some users cited improvement needed for the functionality for adding new cars due to the fact that users could add cars that do not exist. It may be beneficial to add some sort of car model validation to the system in the future.



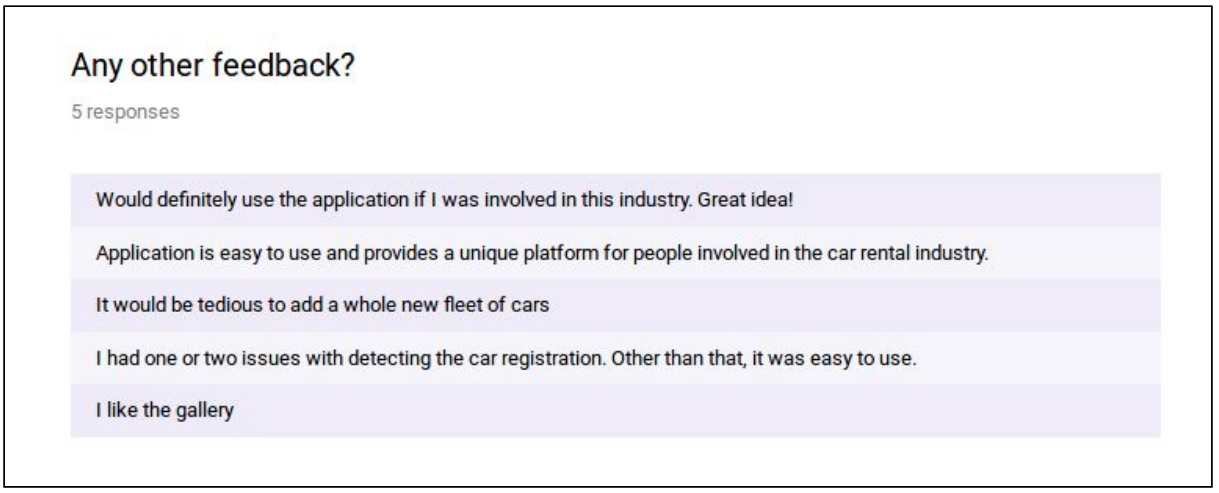
When asked whether they felt the system would improve productivity for the workers of rental companies, all of the users involved in testing the system agreed that it would. In addition, when asked about the look and feel of the application the majority of user felt that the application was designed well and easy to use.



Users who felt the application could be improved left a number of comments throughout the survey indicating where they felt the application could be improved. In general though the feedback was positive.



Some of the additional feedback left by user can be seen below. In general, these comments are positive. However, one user ran into an issue with the car registration being detected incorrectly. Another user made note of how adding a whole fleet of new cars to the application may be tedious, this led to the inclusion of automatic car addition in the future work section of the technical manual.



While performing user testing, no users experienced any major bugs which led to the system becoming unusable. Any bugs which were encountered were either minor or aesthetic in nature, most of which required only minor changes to fix. Some issues encountered still remain, such as the difficulties with the car registration detection. These will be covered in the final section.

8 Known Issues / Bugs Discovered

Issue	Notes
Incorrect Car Registration Recognition	While testing the enclosure's car registration detection, I discovered that the OCR engine would occasionally read the registration plate of a car incorrectly. For example, it may see the number 1 in a registration plate as the letter I. To combat this, the system validates the parsed registration plate to ensure that it meets the Irish registration plate format and is present in the damage analysis system
Overheating of the Raspberry Pi While Performing OCR	While performing batch real world testing with the model, the Raspberry Pi began to overheat. While this was a minor problem in the scope of this project, it would become a much larger issue in production systems.
Incorrect Damage Analysis Predictions	The damage analysis model achieved an accuracy of ~80%, however, while testing in the real world, I discovered that the model can perform poorly depending on the setting of the images (lighting, background noise, etc). In a production system, this can be combated by the use of a controlled lighting system for the enclosure. A solution to combat this in place in the current system is to randomly assign ~10% of cars to be evaluated by a rental employee regardless of the damage analysis model's prediction.
Validation of User-Created Cars	Through the questionnaire as part of user testing, some testers stated how they felt that the system should validate the user's choice of make and model for a car. This could be achieved by having a user choose a make and model from a searchable dropdown rather than an input field and would reduce the likelihood of incorrect or misspelled cars being entered into the system.