To do (1) **To do** (2)

Calculation of Mountain Bike Suspension Settings through Image Analysis

Joe Barrett - 40117680

Submitted in partial fulfilment of the requirements of Edinburgh Napier University for the Degree of BEng (Hons) Software Engineering

School of Computing

04/04/2017

Authorship Declaration

I, Joe Barrett, confirm that this dissertation and the work presented in it are my own achievement.

Where I have consulted the published work of others is always clearly attributed.

Where I have quoted from the work of others, the source is always given. With the exception of such quotations this dissertation is entirely my own work.

I have acknowledged all main sources of help.

If my research follows on from previous work or is part of a larger collaborative research project I have made clear exactly what was done by others and what I have contributed myself.

I have read and understand the penalties of academic misconduct.

I also confirm that I have obtained informed consent from all people I have involved in the work in this dissertation following the school's ethical guidelines.

Signed:

Date:

Matriculation Number:

Data Protection Declaration

Under the 1998 Data Protection Act, The University cannot disclose your grade to an unauthorised person. However, other students benefit from studying dissertations that have their grades attached.

Please sign your name below one of the options below to state your preference.

The University may make this dissertation, with indicative grade, available to others.

The University may make this dissertation available to others, but the grade may not be disclosed.

The University may not make this dissertation available to others.

Acknowledgements

To do (3)

Abstract

To do (4)

Contents

1 Introduction **1**
1.1 Context . . . 1
1.2 Background . . . 1
1.3 Aims and Objectives . . . 2
1.4 Dissertation Structure . . . 2

2 Literature Review **4**
2.1 Mountain Bike Suspension Concepts . . . 4
2.1.1 Travel and Stroke . . . 4
2.1.2 Front Suspension . . . 4
2.1.3 Rear Suspension . . . 5
2.1.4 Sag . . . 7
2.1.5 Damping . . . 8
2.1.6 Optimal Setup . . . 9
2.2 Image Analysis . . . 10
2.2.1 Digital Camera Operation . . . 10
2.2.2 Lighting Conditions . . . 11
2.2.3 Usages . . . 11
2.2.4 Image Analysis Techniques . . . 13
2.3 Image Analysis in Sports Science . . . 16
2.4 Image Analysis for Optimizing Mountain Bike Suspension . . . 17
2.5 Conclusion . . . 17

3 Methodology **18**
3.1 Introduction . . . 18
3.2 Literature Review . . . 18
3.2.1 Source Selection . . . 18
3.3 Platform . . . 19
3.3.1 OpenCV . . . 19
3.3.2 Experimentation . . . 19
3.4 Source Code . . . 21
3.4.1 Pythonic Coding . . . 21
3.4.2 Naming . . . 21
3.4.3 Object Orientation . . . 21
3.5 Testing . . . 22
3.5.1 Unit Testing . . . 22
3.5.2 Project Testing Scope . . . 23
3.6 Project Management . . . 23
3.6.1 Agile Development . . . 23
3.6.2 Organic Development . . . 25
3.6.3 Version Control . . . 25
3.6.4 Milestones . . . 26
3.6.5 Threshold . . . 26
3.6.6 Gantt Chart . . . 26
3.7 Evaluation . . . 27
3.7.1 Validation . . . 27

3.7.2 Reliability and Accuracy 27
3.7.3 Comparison to Alternatives 28
3.7.4 Professional Opinion 28

4 Results **29**
4.1 Deviation from Plan 29
4.1.1 Development Approach 29
4.1.2 Use of EXIF Data and Reference Point 29
4.1.3 Colour Quantification 30
4.1.4 Dynamic Measurement Limits 30
4.1.5 Reference Point Measurement Method 31
4.1.6 Pressure Calculation 32
4.2 Application 33
4.2.1 Images 33
4.2.2 Arguments 34
4.2.3 Processing 35
4.3 Evaluation 38
4.3.1 Validation 38
4.3.2 Reliability and Accuracy 39
4.3.3 Comparison to Alternative Methods 40
4.3.4 Professional Opinion 41

5 Conclusions **42**
5.1 Meeting Aims 42
5.1.1 Aim 1 42
5.1.2 Aim 2 42
5.1.3 Aim 3 42
5.1.4 Aim 4 42
5.2 Comparison to other Products 42
5.3 Future Work 42
5.4 Self Appraisal 42

References **43**

A Android Experiments **48**
A.1 Table of Android Experiments 48
A.2 Hello CV 49
A.3 15tile 50
A.4 BLOB Analysis 58
A.5 Face Detection 63

B Python Experiments **69**
B.1 Table of Python Experiments 69
B.2 find_game.py 70
B.3 thresholding.py 70
B.4 img_ops.py 70
B.5 distance_to_camera.py 71

C Development Log **72**

D EXIF Extraction Code **72**

E Commit Log **72**

F Photos **72**

G Source Code **72**
G.1 main.py 72
G.2 image_processor.py 74
G.3 pressure_calculator.py 76
G.4 test_image_processor.py 77
G.5 test_pressure_calculator.py 79

H IPO **79**

I Week 9 Report **79**

List of Tables

1 Table of common suspension travels and intended disciplines 4
2 Table of pixel data showing an edge 14
3 Table of application arguments 34
4 Table of uncertainty calculation results 39
5 Stages of manual sag setting process 40

List of Figures

1 Hardtail and full suspension mountain bikes (Giant Manufacturing Co. Ltd., 2017a, 2017b) . . . 1
2 Diagram showing travel and stroke on a full suspension bike . . . 5
3 Leverage curves of three modern suspension designs . . . 6
4 Full suspension frame comparison . . . 6
5 Maestro suspension . . . 7
6 Diagram of shock states indicating sag . . . 8
7 A spring and mass system . . . 9
8 Diagram of camera and lens operation (Moeslund, 2012) . . . 10
10 Uses of image analysis . . . 12
11 A copy of Figure 1 with thresholding applied . . . 13
12 Edge detection applied to an image for number plate recognition . . . 14
13 Contact Instrument Calibration Targets on Mars Rover Curiosity (NASA JPL, 2012) . . . 16
14 Normal image (top) versus quantified colours (bottom) . . . 30
15 Red O-ring found using `findContours` (left) with `boundingBox` applied (right) . . . 31
16 Black o-ring found . . . 31
17 Reference point found using HoughCircles . . . 32
18 Reference point found using findContours (left) with boundingCircle applied (right) . . . 32
19 Sag measurements for standard air shock (left) and high volume air shock (right) . . . 33
20 Images suitable for processing . . . 34
21 Processed images for finding reference point . . . 35
22 Locating process for black o-ring . . . 36
24 Example of linear equation plot . . . 37
25 Unit test results . . . 38
26 Test coverage results . . . 39

List of Listings

1 An example of Python code 21
2 Examples of bad naming (left) and proper naming (right) taken from Clean Code (Martin, 2009) 21

1 Introduction

1.1 Context

The suspension on a mountain bike plays a vital part in the rider's performance, comfort and overall enjoyment of the sport. With some suspension units costing upwards of £1000 it is vital that they are setup to function correctly. The objective of this dissertation is to examine how image analysis can be used in the set-up of mountain bike suspension, and produce a prototype application that helps beginner and intermediate riders to carry out an initial set-up.

1.2 Background

A survey carried out by the International Mountain Bike Association found that the average price had paid for a mountain bike was €2546 (£2206) (IMBA Europe, 2015). Starting at approximately £1000 (Giant Manufacturing Co. Ltd., 2017c), enthusiast level mountain bikes can be purchased with suspension for both the front and rear wheels, known as Full Suspension (FS) bikes whereas Hard Tail (HT) bikes only have front suspension units; this difference can be seen in Figure 1. The price of higher specification mountain bikes can run to many thousands of pounds, but even entry level models are equipped with adjustable suspension units that need to be correctly configured to the rider's weight and the characteristics of the terrain.

Figure 1: Hardtail and full suspension mountain bikes (Giant Manufacturing Co. Ltd., 2017a, 2017b)

It has been proven that using a FS bike as opposed to a HT gives the rider a measurable performance advantage (Titlestad, Fairlie-Clarke, Davie, Whittaker, & Grant, 2003). However, if the suspension fork and/or rear shock have not been correctly configured it can be detrimental to the rider's performance and potentially lead to injury. For example, if a shock has too little rebound damping set and the rider goes off a jump, the excessive speed at which the rear of the bike extends can create forwards rotation, causing the rider to go over the handlebars of the bike. However many enthusiasts lack the specialist knowledge required to optimize the set-up of their bike's suspension units and as a consequence compromise their performance and riding experience.

Additionally, an incorrect suspension setup can cause excessive wear and tear on the bike's frame and components. For instance, suspension which is set too soft will cause "bottoming out" where the unit reaches the end of its available range so that excess forces are transferred to the frame causing stress and potentially dramatic fracture that could result in injury. Conversely, suspension that is set too hard forces energy that would normally be absorbed to the wheels resulting in denting and warping of the rims.

Many bicycle retailers will configure the suspension of a newly purchased mountain bike for the customer at the time of collection. Most of the time this will be enough to avoid incident but as the customer is unlikely to be wearing weighty protection equipment and accessories such as helmet, body armour and hydration pack, this initial setup is rarely accurate. Furthermore, with some manufacturers choosing direct sales over local retailers (Harker, 2010; Staff, 2015), even this rudimentary setup can be omitted altogether.

Since the birth of the modern smartphone in 2007, brought about by the first generation Apple® iPhone® and introduction of the Android™ mobile operating system, the use of mobile computing in everyday life has grown rapidly. Google™ stated that there were approximately 1.4 billion active Android users worldwide in 2015 (Callaham, 2015).

The introduction of activity tracking devices and mobile applications such as FitBit (Diaz et al., 2015) and Strava (West, 2015) and their growing popularity (Formosa, 2012) shows that many individuals like to use their smartphones to aid or augment their performance and enjoyment of their chosen hobby or sport. On the back of this increase, a number of companies have launched small devices (Aston, 2016; Hwang, 2016) and mobile applications (Benedict, 2012) designed to help riders setup and fine tune their own suspension, either at home or while out on a ride. Each of the devices produced cost more than £100 and require the suspension to have an initial set up in place and the only mobile application capable of producing an initial setup is tied to a single suspension manufacturer.

1.3 Aims and Objectives

The aim of this project is to create a prototype application capable of providing the user with a suggested suspension setup using image analysis techniques. This will be achieved by meeting the following objectives:

- Complete a literature review of mountain bike suspension and image analysis techniques including how image analysis is currently used in sports science.
- Implement the prototype application using identified and researched methods
- Evaluate the success and appropriateness of the produced application
- Present conclusions about the project's successes and downfalls

1.4 Dissertation Structure

This dissertation will document the project by taking the following structure:

1 **Introduction** - Outlines the context of the project and states the major aims and objectives

2 **Literature Review** - Presents a literature review to further describe the project context and provide information on mountain bike suspension concepts and image analysis techniques

3 **Methodology** - Discusses the methodologies available which could be used to complete the project and their advantages

4 **Results** - Describes the outcome of the project including any deviations from the plan, the operation of the application, and a critical evaluation of the application

5 **Conclusion** - Presents conclusions and learnings from completion of the project including potential future work and a self appraisal

2 Literature Review

The aim of this project is to produce an application that makes use of image analysis techniques to provide the user with a suggested set-up for their suspension units. This literature review will identify the intricacies involved in correctly configuring a rear suspension unit and how these make it difficult for riders to do it themselves. It will then go on to highlight how image analysis is carried out and identify the various techniques that are relevant to this project. This review will shed light on the problem at hand and help identify a viable solution.

2.1 Mountain Bike Suspension Concepts

The purpose of suspension on a mountain bike is to absorb the energy created from riding over features, such as bumps and rough terrain encountered along a trail, improving comfort for the rider and allowing them to go faster by maintaining better contact between the tires and the ground. This requires the use of a spring and damper, collectively known as a shock absorber, which allows the wheel to move away from the feature on contact and make a controlled return once it has been passed.

2.1.1 Travel and Stroke

Travel is the distance which the bike's fork or frame allow the wheel to move in an upward direction while stroke is the distance that the shock absorber can compress before it bottoms out. Travel is measured in millimetres or inches and can range from 80mm to 210mm or 4in to 9in. Bikes designed for different disciplines require differing amounts of travel with those designed for cross-country riding typically requiring less travel than those designed for more aggressive disciplines such as downhill racing. See Table 1 for typical specifications.

Table 1: Table of common suspension travels and intended disciplines

Travel (mm)	Cross Country	Trail	Enduro	Downhill
80				
100				
120				
140				
160				
180				
+200				

2.1.2 Front Suspension

Front suspension commonly employs a linear telescoping shock absorber, known as a fork due to it's dual sided construction. On nearly all suspension forks the stroke is 1:1 with the potential travel of the wheel. Front suspension is found on all FS and HT bikes.

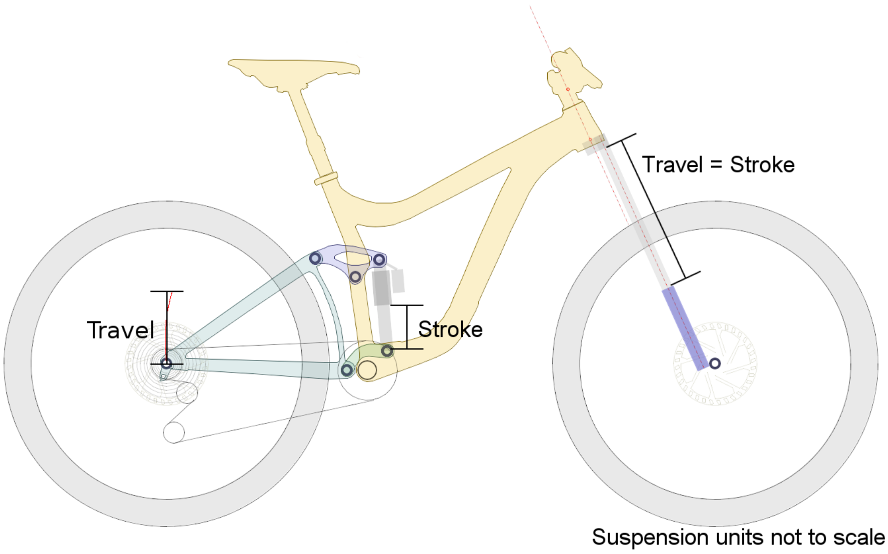


Figure 2: Diagram showing travel and stroke on a full suspension bike[1]

2.1.3 Rear Suspension

Rear suspension uses a shock absorber that is much shorter than a fork so it cannot operate on a 1:1 ratio while still allowing the full amount of travel required. Full suspension frames incorporate one or more pivot points and linkages which allow the wheel to move and act as multipliers for the suspension. Rear ratios are expressed as n:1 where n is the average distance the rear wheel moves for every 1mm the shock compresses throughout its stroke; the average is used as the leverage ratio changes across the compression cycle.

The difference between front and rear stroke and travel can be seen in Figure 2 which shows the separation of rear wheel travel from the stroke of the shock absorber. Although manufacturers design their rear suspension differently, the rear wheel always rotates around the main pivot, (or in some cases a virtual pivot), as opposed to moving linearly as front forks do. As a result, the frame behaves differently through its travel, depending on the number and location of pivot points and the type of shock that it is being used.

Because of this the average ratio is normally dismissed in favour of a leverage curve that plots the ratio n:1 throughout the compression cycle. Figure 3 shows the leverage curves of three modern suspension designs. Each of these designs has between 150mm and 170mm of travel and uses the 27.5 inch wheel size. However, it is evident that varying the location of pivot points produces suspension with drastically different characteristics.

The Virtual Pivot Point (VPP) design of the Giant Reign (shown blue) has an initial falling rate, meaning the shock can be compressed easily, but slows down and even rises slightly towards the end of its travel as seen in Figure 3. This means the suspension will feel soft most of the time but stiffens as compression increases. This is emphasised by the Horst link system of the Lapierre Spicy (shown magenta) where the leverage curve rises significantly towards the end of its travel. In contrast, the curve of

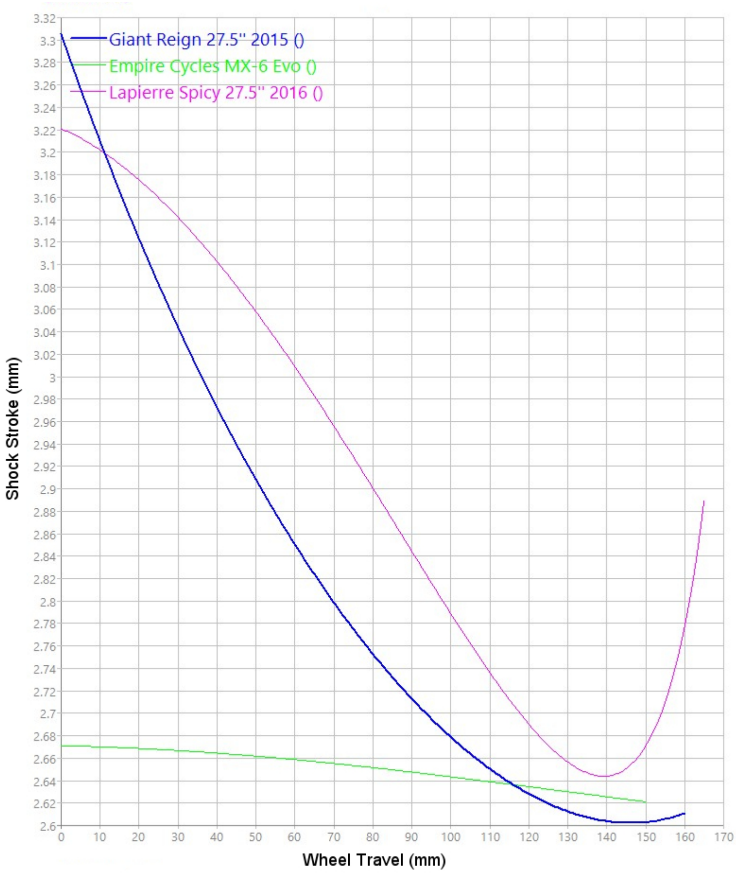
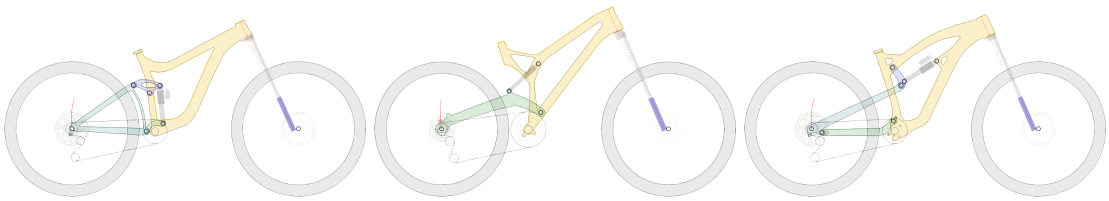


Figure 3: Leverage curves of three modern suspension designs

the single pivot Empire MX-6 Evo (shown green) is effectively linear. This is due to the MX-6 having only one pivot and swinging arm, as opposed to the multiple pivots and linkages of the VPP and horst link designs, so there is an almost direct input from the rear wheel to the shock. The physical differences in each of these frames can be seen in Figure 4.



(a) Giant Reign (b) Empire MX6 Evo (c) Lapierre Spicy

Figure 4: Full suspension frame comparison

For this project two bikes will be used to develop and test the application. The first is a 2015 Giant Reign, shown on figure 3 in blue, as it will be constantly available throughout the project. The frame uses Giant's Maestro™suspension system which is a variation of VPP. Like all VPP systems Maestro uses two links, an upper and lower, to create a virtual main pivot point. However unlike other VPP systems, Maestro creates its virtual pivot as close to the rear of the frame as possible as shown by the red circle in Figure 5. The second will be a 2011 Orange 5 which has the same suspension design as the Empire MX6 Evo in Figure 3. This bike was selected as it has a completely different design to the Giant Reign and will also be easily accessible throughout the project.

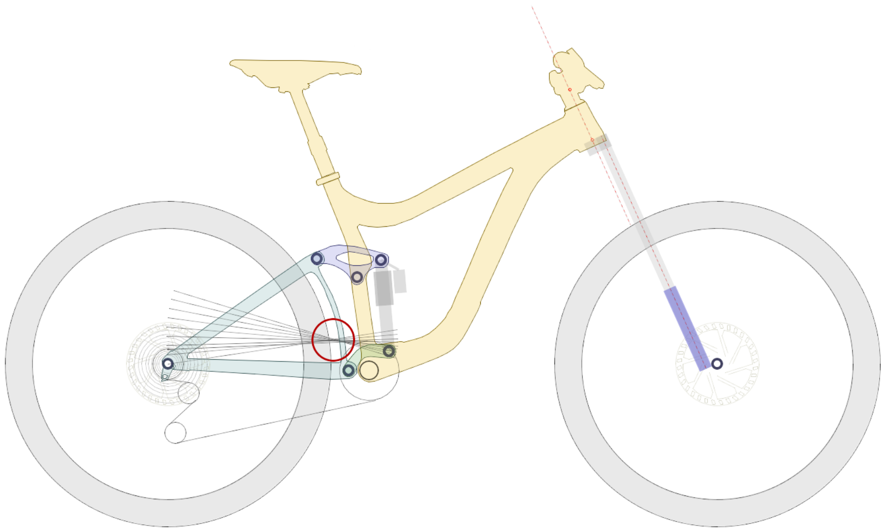


Figure 5: Maestro suspension

Although rear suspension designs are complex, neither the average rider nor professionals, such as mechanics working in the industry, necessarily need this specialist knowledge. Leverage curves are predominantly used by designers to determine how a particular suspension unit will behave when researching and designing new frames (Cane Creek, 2016). Though having an understanding of leverage curves will help the individual rider to fine tune their suspensions settings themselves, the majority will take little interest in the differences between a single pivot and VPP design, choosing a simpler "set and forget" approach to their suspension.

In the context of this project and the intended target audience for the application, this begs the question of how much information should be provided to the user? Modern human /computer interaction principles aim toward providing information to the user which is relevant and necessary in context (Shneiderman, 2010). Presenting the user with a leverage curve which may require extensive interpretation before they understand the implications within the limitations of a mobile application may prove detrimental to the user experience. For an application which is intended to remove the difficulties of suspension setup and allow for quick and easy production of a basic setup, providing a single sag setting would be preferable over a plethora of of technical data.

2.1.4 Sag

To do (5) Sag is the amount that the suspension sits into its travel when the rider is in a neutral position, as described in Figure 6. It is required so that the suspension has travel available to extend as well as just compress and is calculated based on the rider's weight, available travel, and intended riding style.

The amount of sag depends on the stiffness of the suspension unit which can be adjusted by changing the air pressure of an air spring or replacing the spring and adjusting the preload of a traditional coil shock. Depending on the discipline and the amount of travel the bike has, sag can vary between 15% and 40% of the available travel though it is typically set at between 25% and 35% for the average rider with greater variances only encountered in competition.

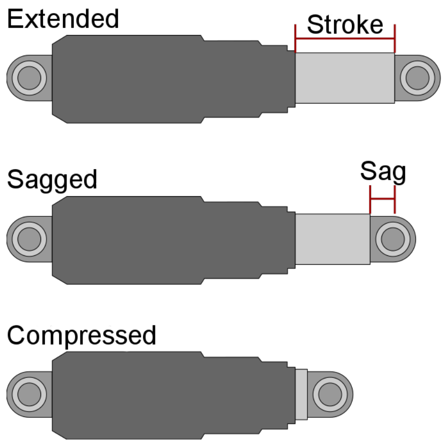


Figure 6: Diagram of shock states indicating sag

Sag is set by first calculating the distance that the shock should sit into its travel by producing the desired percentage of the shock's stroke. For an air shock the manufacturer's recommended pressure is then pumped into the shock, and the rider weights the bike. On air shocks there is a marker O-ring which is pushed down the shock shaft as it is compressed. The position this O-ring ends at can be measured and the shock pressure adjusted accordingly until it sits at the target measurement. For coil shocks, the total length of the shock must be measured while the bike is under load.

2.1.5 Damping

In a spring and mass system (visualised in Figure 7), when the spring is elongated it will exert a returning force on the mass, labelled F_k. According to Hooke's law (Rychlewski, 1984), this force is directly proportionate to the distance which the spring has been pulled. In suspension this force is explained as two separate forces. If an individual were to push down on mountain bike suspension they would feel a resistance, this is known as compression, and when they let go the suspension will return to its neutral state, this is known as rebound. Both of these can be controlled and are referred to as compression damping and rebound damping respectively.

Suspension damping works by forcing oil within the shock absorber through a series of holes in the absorber's damping circuit. Reducing the size or number of holes increases resistance, reducing the speed at which the oil flows through the circuit to increase the damping effect by making compression and rebound slower.

2.1.5.1 Compression Damping This is applied while the shock absorber is being compressed to control the amount of effort required. Increasing damping forces the wheel to stay in contact with the ground for longer making the suspension feel stiffer. However too much compression damping can make the suspension overly stiff so it does not effectively absorb the impact of bumps and rough terrain. Conversely, too little

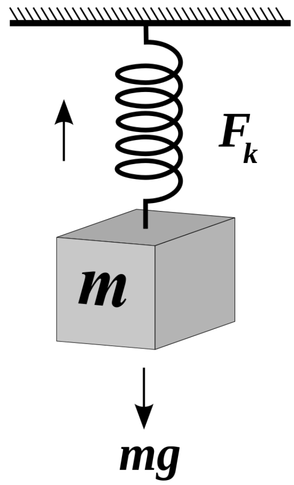


Figure 7: A spring and mass system

compression damping can cause the suspension to ”blow through” all of the available travel prematurely with nothing left to soak up impact when it is most required.

2.1.5.2 Rebound Damping This is used to control the speed at which the shock absorber recovers to its normal riding position after compression. An optimal setting will allow the suspension to track the ground, quickly and smoothly returning to the correct position after a bump or hole is passed.

Too much rebound damping causes the suspension unit to recover slowly and sometimes “pack down” meaning the shock absorber remains compressed for too long leaving the rider fully exposed to the next impact. Too little rebound damping can cause the suspension to “buck” the rider, like a horse, and potentially cause an accident.

2.1.5.3 High and Low Speed Damping The ways of adjusting compression and rebound damping differ by manufacturer and model with better specified bikes having two adjustable speeds for each damping circuit giving four damping settings. High speed adjustments are used in high G-force situations such as riding large jumps or drops where compression needs to be set softer to absorb impacts and rebound set slower so the rider has time to recover without the equilibrium of the bike being upset.

Low speed damping set ups are used against lower G-force movements such as rider weight shifts or long, slow compressions. Optimally compression is set stiffer as this type of feature can use a lot of travel and rebound damping set faster to deal with multiple features in quick succession.

2.1.6 Optimal Setup

Although setups will vary between rider, suspension system and discipline, there are some key principles that all riders should aim to achieve. Sag should be set to an appropriate measurement by adjusting the air pressure on air shocks or spring rating on coil shocks. Compression damping should feel soft and soak up bumps efficiently without excessive bottoming out. Rebound damping should be set to return as fast as possible without bucking the rider, which is normally somewhere in the middle of the two setting available with a slight bias towards the faster option.

Attaining this optimal setup can be difficult for both beginners and intermediate riders as they lack experience and in-depth knowledge of suspension units and may not know how different frames react while being ridden. Knowing which measurements to make and the calculations required to correctly configure sag, compression and rebound settings are typically beyond the capabilities of this level of rider unless they have been previously trained by a professional or investigated the topic in detail for themselves.

2.2 Image Analysis

Image Analysis (IA) is the use of various techniques, such as pattern recognition, geometry calculations, and signal processing, to extract information from digital images. Image processing is the application of various processes on an image to change or enhance its appearance. In practice, the processing stage normally comes before the analysis stage as a way of simplifying the image prior to analysis in order to maximize the likelihood of obtaining usable data.

2.2.1 Digital Camera Operation

A digital camera operates by capturing visible light reflected by objects onto the camera's sensor. The light must travel from the object through a convex focusing lens which refracts the light onto a corresponding point on the sensor.

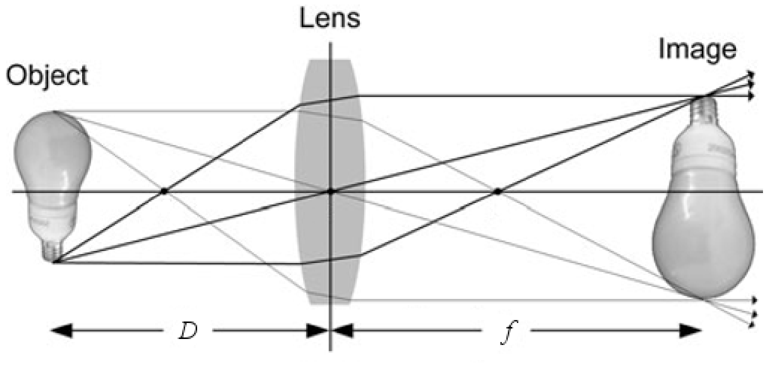


Figure 8: Diagram of camera and lens operation (Moeslund, 2012)

The distance between where light enters the lens and the point at which it is no longer diffused is known as the focal length, marked f in figure 8. This point can be adjusted by changing the optical characteristics of the lens so that the refracted light hits the sensor to create a sharply focussed image.

A camera's sensor is made up of an array of photosensitive cells capable of collecting light and generating an integer value based on the brightness and colour of the received light. The microprocessor inside the camera takes these values and converts them into the image data, sometimes taking an average of the surrounding values to better understand the light as it was captured. Each of the cells in the camera's sensor equates to a single pixel in the final image. For example, a camera with a sensor made up of 1920 cells across by 1080 down (otherwise known as a 2 megapixel sensor) produces an image 1920 pixels wide and 1080 high. The larger the physical size of the sensor, the more cells it can contain resulting in a higher quality image.

Image processing is the manipulation of these integer values to adjust the visual appearance of the image for either artistic or scientific purposes. Image analysis is the comparison of these values either to their neighbours or in clusters to locate features, produce measurements, or for other purposes within the context of the image.

2.2.2 Lighting Conditions

When capturing images to be analysed, it is important to have good lighting conditions so that none of the required detail is lost (Moeslund, 2012). If the image is captured in unsuitable conditions, such as low light, then this can seriously affect the outcome of subsequent analysis.

Certain image processing techniques, such post-processing images captured as Camera RAW files, can help ameliorate poor lighting conditions though this is not foolproof and it is always better to capture a well-lit image in camera. Figure 9 shows the effects that different lighting angles have on a subject.

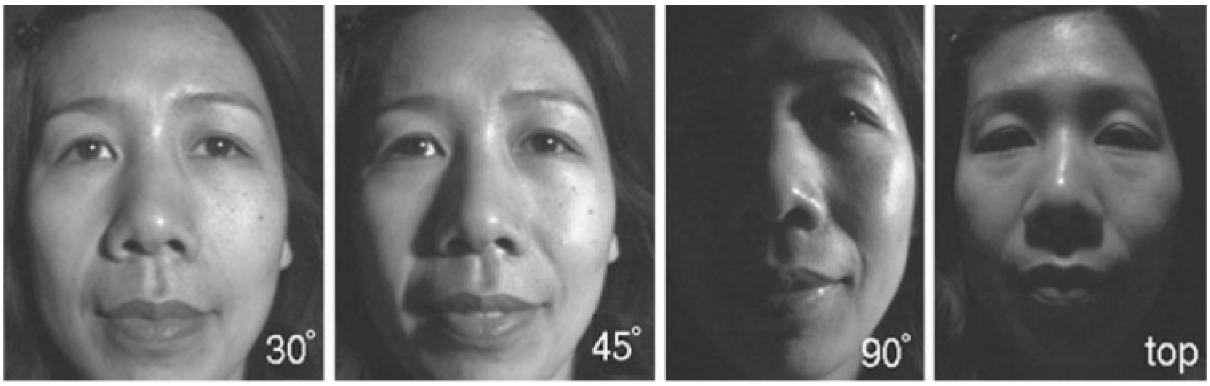


Figure 9: The effects of different lighting conditions on a face (Moeslund, 2012)

It can be seen that the situation which highlights the most detail is when the light source is close to being in front of the subject matter. Such lighting reduces the amount of image processing required before analysing and produces the largest amount of usable data. In the context of this project many smartphones also have an integral flash function which means the user can correctly light their image in sub-optimal lighting conditions where required detail may otherwise be lost.

2.2.3 Usages

Image processing and image analysis have been applied to multiple areas with its value and effectiveness rapidly improving alongside advances in camera technology and computing power. These applications range from facial recognition in social media uploads (Zuckerberg, Sittig, & Marlette, 2011) to the utilisation of satellite imagery to track the changing shape of coastlines (Potter, 2013).

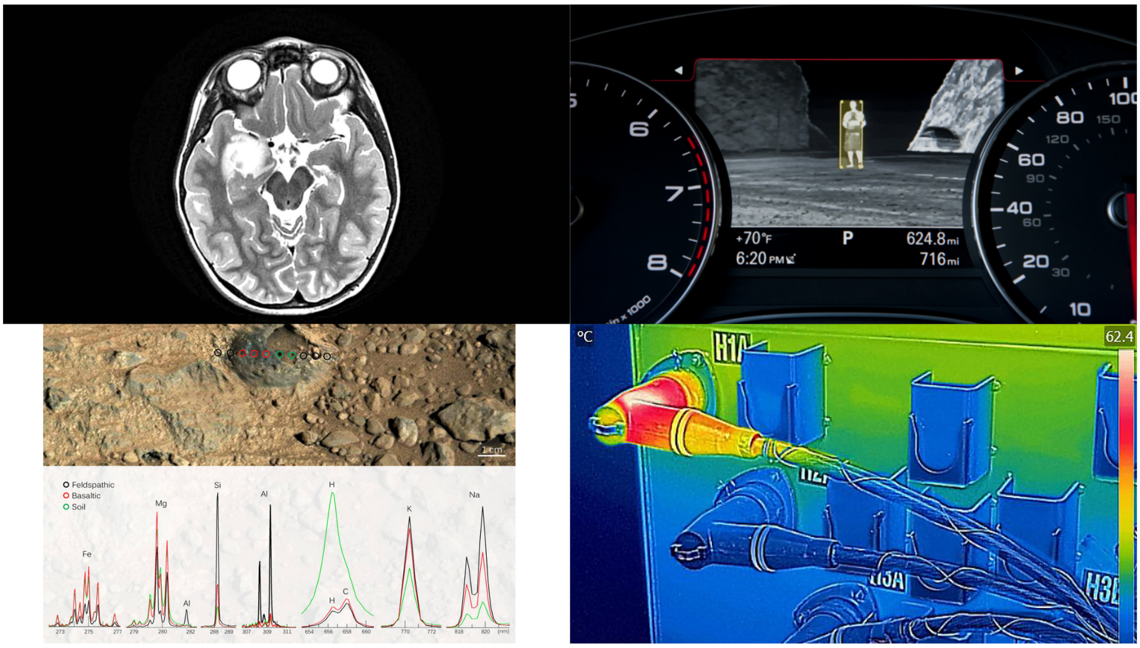


Figure 10: Uses of image analysis, from top left clockwise: An MRI brain scan, automotive night vision with pedestrian recognition, infrared image taken with a smartphone, chemical rock analysis from mars

2.2.3.1 Medical This is arguably one of the most important uses of IA. Advances in medical imaging have reduced costs, diagnosis time and patient recovery time while improving the ability to localise and personalise treatments (European Science Foundation, 2007). Major uses of IA in medical applications are the use of Magnetic Resonance Imaging (MRI) and Computerised Topography Scanning (CT Scan) to create detailed images of the human body and identify illness before obvious symptoms appear. This is shown top left in Figure 10.

2.2.3.2 Transport Image analysis has been included in the consumer automotive market on various models since 2004 when Honda introduced a thermographic night vision camera with automatic pedestrian detection on their Legend model (Honda Motor Co., 2004). Other manufacturers, including Audi, have since introduced similar technology and their implementation can be seen top right in Figure 10. Since this initial use many other vehicle manufacturers have included functionality based on image analysis expanding its use into areas such as the automatic recognition of speed limit signs, lane departure warning systems, and automatic braking systems based on hazard recognition.

2.2.3.3 Engineering The use of image analysis in engineering has helped to create more stable and efficient structures, in bridges and skyscrapers for example, by looking at the materials used in their construction (Masad, Muhunthan, Shashidhar, &

Harman, 1999) and monitoring their stresses and potential areas of weakness (Kim & Kim, 2013). Today advances in mobile computing have allowed engineers to routinely use IA while on site and some application vendors target their products at an engineering sector by improving their durability and integrating features such as infrared imaging (Liszewski, 2016), the output of which can be seen lower right in Figure 10.

2.2.3.4 Space While some industries make use of satellite imagery to monitor changes to our own planet, agencies such as NASA and ESA use image analysis to look at other planets and celestial bodies. The Martian rover, Curiosity, uses multiple cameras for navigation, hazard avoidance, and scientific imaging with the images streamed back to Earth for detailed analysis. Major uses of such extraterrestrial images include the identification of geological formations and their likely composition (Blake et al., 2013; Garvin, Malin, & Minitti, 2014) and the location and identification of chemicals using the analysis tools such as ”ChemCam” (Schröder et al., 2015), which can be seen lower left in Figure 10.

2.2.4 Image Analysis Techniques

There are a number of analysis techniques which can be applied to imagery to extract information. Perhaps the most common is analysing the pixels in an image to identify discontinuities in their data values which suggests the edge of an object, structure or land mass. Such functionality is readily available on desktop image processing packages designed for amateur photographers (Adobe, 2017). But scientific and engineering applications of edge detection provide far great levels of sensitivity helping isolate objects that can be subsequently measured with high level of accuracy (MatLab, 2017).

2.2.4.1 Thresholding Image thresholding is the most basic form of image segmentation (Haralock & Shapiro, 1991). At its simplest this would entail taking a greyscale image, examining the intensity of each pixel and setting those below a certain threshold to black and those above it to white as shown in Figure 11 below. By thresholding an image, objects lose detail but their silhouettes become clearer making them more visible to object detection techniques.

Figure 11: A copy of Figure 1 with thresholding applied

2.2.4.2 Edge Detection This is the application of mathematical algorithms to locate and highlight the edges of features in an image. There are multiple algorithms which can be used for edge detection, including Sobel, Roberts, Canny, and fuzzy logic, although all utilise the same basic concept of comparing the data values of adjacent pixels to find discontinuities from one value to another.

Table 2: Table of pixel data showing an edge

5	7	6	4	152	148	149

Table 2 represents possible pixel values of an edge indicated by the large difference between 4 and 152. The applied algorithm will pick up this discontinuity which will be highlighted on the resulting image. A common application for edge detection is text recognition such as in automatic number plate recognition (ANPR) (Ahmad, Boufama, Habashi, Anderson, & Elamsy, 2015) where unwanted background data is automatically excluded using thresholding techniques to highlight the block shapes of the vehicle registration plate. Figure 12 shows an image which has had edge detection applied with the number plate of the car clearly visible.

Figure 12: Edge detection applied to an image for number plate recognition

2.2.4.3 Hough Transform **To do** (6) Originally created in 1962 by Paul Hough, the Hough transform methods offer techniques for the discovery of imperfect objects in images. The original algorithms were created for the detection of lines but have since been improved and adapted to find circles, ellipses, and other basic shapes.

While edge detection is suitable for highlighting critical points in an image, it can be problematic if the base image is not well lit or sharply focussed causing an edge detector to produce imperfect lines. Such artefacts can make rudimentary feature detection techniques difficult but the Hough Transformation techniques circumvent such issues by considering points as a part of an object through a voting procedure.

Linear Hough Transform To locate straight lines in an image, the linear Hough transform utilises Hesse normal form $r = xcos\theta + ysin\theta$. Lines, or planes, in an image are expressed as pairs of (r, θ)

Circular Hough Transform

2.2.4.4 Taking Measurements To measure the dimensions of an object in an image, certain data about the camera and its location are required. By using digital imagery, the majority of this information is automatically provided as each image contains metadata or EXchangeable Image Format (EXIF) data which includes information such as camera manufacturer, focal length, image size, and location if available.

$$H_o = \left(\frac{f \times \left(\frac{H_i}{H_s}\right)}{D - f} \right) \times H_c \tag{1}$$

where

- f is the focal length of the camera lens
- D is the distance to the object
- H_o is the height of the object
- H_i is the height of the image in pixels
- H_s is the height of the camera sensor
- H_c is the height of the camera from the ground

The process to calculate the height of an object in an image is shown in Equation 1. Necessary data, such as f, H_i, and H_s can be acquired from EXIF data. However D and H_c are not collected by the camera and must be measured. When dealing with image analysis this means that this data has to be collected at the time the image is taken.

However it is not necessary to collect data for H_o - the distance of the object being measured from the ground - as the size of the object is in direct proportion to its size on the camera's sensor so is not required. To test this equation, data was created from known object heights which was then re-used to see if the calculated height was equal to the actual height. For example:**To do** (7)

To circumvent the need to collect data for H_c - the height of the camera -, a reference object of a known size can be included in the image as this allows a comparison to be made between the height of the object to be measured and the reference object. The previously mentioned Mars rover, Curiosity, carries a United States penny as a reference object so that its cameras can be precisely calibrated using reference charts as shown in Figure 13.
However, the reference object must be located in the image using a technique such as **To do** (8) BLOB analysis or Hough transforms. Once the object is found, its width or height in pixels in the image is collected and compared against its actual size to give a ratio for applying to the pixel count of the object to calculate its actual height in real world. For example, if reference object that is 2mm wide measures 200 pixels across in an image then it can be concluded that 100 pixels in the image corresponds to 1mm in the real world. Knowing this ratio, any other objects that can be measured in pixels can have the real world measurement estimated as long as they are close to the same

Figure 13: Contact Instrument Calibration Targets on Mars Rover Curiosity (NASA JPL, 2012)

distance from the camera as the reference object.

This process will be applied in some manner in this project. Either a reference object will be used or measurements for H_c (the height of the camera from the ground) will be taken at the time the image is captured as knowing these measurements is vital to achieving reliable results that will ensure that suspension units are correctly configured as previously discussed. But while incorporating Equation 1 in an application will be relatively simple, collecting the necessary data without too much user interaction may be more difficult.

2.3 Image Analysis in Sports Science

The benefits of image analysis in sports science are most prevalent in the area of biomechanics. In most sports , an individual's performance depends on a combination of physical fitness and skills. In certain sports, such as motorsports, equipment is undeniably important although unless "driver-athletes" can cope with the stresses and strains of race conditions they are unlikely to achieve success regardless of the technical superiority of their vehicle (Klarica, 2001).

In such sports, the performances of cars, motorcycles, and even mountain bikes are all predictable and measurable. They have all been designed and manufactured to be as fast as possible within the rules laid down by the sport's governing body so it is important to continually analyse and fine tune their set – up by changing tires or adjusting suspension to seeking out the margin gains that make the difference between gaining a place on the podium or not. As a result, performance cars and bikes are fitted with an array of sensors that constantly monitor critical aspects of performance (Segers, 2008).

Fitting sensors to humans without impairing their mobility is not quite so simple. To provide the best testing ground for fitness and technique, the participant should be unhindered and able to perform tasks without data capturing equipment getting in their way. Image analysis can play a large part in this as utilising various techniques can allow for stable and repeatable test situations while also providing a platform for reviewing the captured images. A study into bowling techniques in cricket used a mix of manual point picking and automated measuring from images to produce data such as angle and speed of bowling deliveries as well as ball spin (Cork, Justham, & West, 2012). While the dataset was limited due to conflicts with the players' training schedules, the results proved useful and were subsequently replicated in a pitching machine so that batsmen practised against more lifelike deliveries.

A second study on the use of IA in showjumping (Wejer, Lendo, & Lewczuk, 2013) made use of techniques similar to those used by Cook, Justham, and West. In this instance, the angles of the horse's limbs were recorded and compared over a period of four months to analyse whether different training techniques delivered measurable improvements. Here, passive image analysis was chosen as the preferred technique as attaching sensors to the horse could have frightened the animal and almost certainly caused it perform below par.

2.4 Image Analysis for Optimizing Mountain Bike Suspension

Though image analysis has been used for calculations in many engineering and sports applications (Kim & Kim, 2013; Masad et al., 1999), it is relatively unused with mountain bike suspension with only the specialists at Fox Racing Shox using the technology (Benedict, 2012). The mobile application that Fox has locks to the forks and shocks that the company manufacture although the image analysis techniques they use are adaptable enough to be used on suspension units from other manufacturers.

By harnessing the image capturing and computing power of modern smartphones, image analysis can be applied to mountain bike suspension to produce a simple method of calculating a baseline setup for any suspension unit. The measurements and calculations required can be removed from the user's responsibility to create a simple and efficient method of generating a safe and reliable suspension setup.

2.5 Conclusion

By identifying the key aspects and settings relating to mountain bike suspension and the common image analysis techniques, this literature review has highlighted the processes that this project can utilise to achieve its aims. By using this knowledge, the following sections will identify distinct methods which the project could use and cover the results of this.

3 Methodology

3.1 Introduction

This chapter gives an outline of the methodologies used to complete the work identified in the previous chapters as well as the reasons for using these methods and rejecting alternatives that could have been used. The effectiveness of these methods determines the success of the project so those chosen need to be both reliable and manageable. A technical approach was identified for the most appropriate way to produce a solution to the problem identified and a project management approach was decided on so that the project could remain on track and meet the required deadline.

3.2 Literature Review

The purpose of the literature review presented in section 2 of this dissertation is to outline, investigate, and clarify the subject areas with which this project is involved. This aids both the reader and author in understanding these subject areas and helps explain to how the project presents a viable solution to the problem area.

There are numerous forms a literature review can take although not all have the same aims. A traditional or narrative review can be carried out to critique a body of literature and potentially locate inconsistencies (Adams, Khan, Raeside, & White, 2007). This type of review is normally used when the research question is well defined. Alternatively a systematic literature review can be undertaken although this requires a more rigorous process and takes more time. Systematic reviews aim to locate previous studies that are within the same or similar subject areas and gain insight to expedite answers to the questions presented by the research (Kitchenham et al., 2009).

For this project a combination of both methods has been used. A systematic approach was applied to investigate how image analysis is used and ascertain which particular techniques were relevant to the use of IA in the context of mountain bike suspension. A traditional method has been applied to examine the uses of image analysis in sports science to determine whether previously used techniques would transfer to mountain bike suspension and provide some idea of how effective they could be.

3.2.1 Source Selection

Regardless of which types of review was chosen, it was necessary to adopt a methodological approach to identify sources relevant to the project. Multiple services are available for viewing academic papers online and for this project the Edinburgh Napier University library and Google Scholar were utilised as between them they provide an expansive library to select from using a variety of advanced search filters.

As the application of advanced technologies to mountain bike suspension is in its infancy and commercially driven, much of the research is undertaken by the manufacturers themselves so there were no academic sources to use. As a result, numerous multiple mountain bike websites and blogs have been cited. In a subject area where the research is more established this would be frowned upon as such sources tends to be biased and often unproven. However it was entirely unavoidable.To overcome author bias, multiple sources have been cited wherever possible as a way of providing different viewpoints on the same topic.

To determine whether a source is suitable requires a methodical and critical approach. Even before reading a paper or article, it is possible to decide whether the subject matter and research is up to date by referring to its publication date. Such an approach was adopted for this project in order to identify the most recent research available. As a second step, online libraries provide metrics about the number of times a paper has been cited elsewhere; the inference being that papers that have been cited most are both more relevant and trustworthy. However, applying this rule of thumb needed care as some papers are cited so frequently purely because they are the most controversial. Finally when reading a paper, the abstract should be read first followed by the conclusions. This takes little time and taken together, the abstract and conclusions provide a good indication of whether the paper contains relevant and reliable research. Only then should the body of the paper be read in its entirety.

3.3 Platform

As a proof of concept, the final product of this software project could take a variety of forms. In line with current products on the market an application for Android could be produced which would demonstrate the capabilities of image analysis on a mobile device. Alternatively, the image analysis algorithm can be produced in the Python programming language as a script creating a simpler prototype but clarifying the solution that is the core of the problem.

3.3.1 OpenCV

OpenCV is an open-source computer vision library created for a variety of platforms. Originally released by Intel in 1999 (Bradski & Kaehler, 2008) the library was intended as a research project to aid in CPU intensive visual applications. In 2012 the library was taken over by OpenCV.org (OpenCV.org, 2017), a non-profit organisation who provide support and documentation. Although the library is written in C and C++, modules are available for Python, Fortran, and the Android platform.

This project will use OpenCV as it is widely accepted as the best, free computer vision library available and provides functions for the various processes this project requires. Additionally it is well supported and documented which will aid in the development process.

3.3.2 Experimentation

To further understand what may be required to produce both the Android and Python based approaches and aid in deciding which method to use some basic experiments were carried out to compare the two. For the Android experiments some example applications which are bundled with the OpenCV library were hand copied and run on a mobile device. This allowed their output and functionality to be seen while simultaneously gaining experience in using OpenCV on Android. For the Python based approach, tutorials from the www.pyimagesearch.com website created by Adrian Rosebrock (Rosebrock, 2017) and the www.pythonprogramming.net website (Pythonprogramming, 2016) creators name unknown were followed and run on a desktop computer.

Appendix A shows the experiments which were carried out on the Android platform. It should be noted that, while all applications do not show compilation errors and were adjusted to work with the updated version of Android which the device was using, only two of the four experiments worked successfully. Although a stack trace of the errors was produced they do not explain the cause of the error. This is due to how OpenCV works on the Android system.

Alongside the Android Software Development Kit (SDK), Google provide a Native Development Kit (NDK) to allow modules which were not originally written in Java to be run on Android devices. The NDK understands various programming languages and applied a Java wrapper around them that is capable of extracting their functionality. This must be applied when using OpenCV on Android as it is written in C++. An artefact of using the NDK is that it cannot convert the stack trace produced by errors in the C++ module and following research into this issue it was found that this is difficult to rectify.

The two working Android experiments do not operate as expected either. Appendix A shows the issues with each that were encountered in the test. Although steps were taken to rectify the orientation in the Hello CV experiment and the full-screen issue in both, any fix which was applied was not accepted by the system. Research and debugging of these faults did not rectify these issues to produce an acceptable outcome. Appendix B shows the experiments that were carried out using Python. These were much more successful than the Android experiments as they are all functioned and worked exactly as anticipated.

Appendices A and B show a comparison between the two methods including the number of files and lines of code required to create the program as well as the relevant source code for each program. There is a clear difference between the two methods. The Android method taking an average of 295 lines of code over 3 files to produce arguably less complex and less functional applications than is produced by Python's 21 lines of code over 1 file.

Due to the difficulties encountered when using OpenCV on Android, the decision has been made to produce a proof of concept for the image analysis in Python which also benefits from being an operating system that is already a proven platform for mobile applications. This will allow the project to maintain focus on the analysis and algorithmic side of the solution as opposed to the portability. The decision also allows for more time to be spent making the program functionally stable which will serve as a better demonstration of the solution.

3.4 Source Code

3.4.1 Pythonic Coding

A metric of software quality is the conciseness and descriptiveness of the source code. The aim of the Python programming language is to complete the same tasks as other object oriented languages but in fewer lines and in a more readable manner (Kuhlman, 2009). By removing brackets and instead using indentation to define the boundaries of classes, functions, and conditionals as well as using worded operators, Python creates source code which reads like a list of instructions for people rather than computers; this is demonstrated in Listing 1. The source code for this project will be written in a pythonic way where it makes sense to do so.

```
long_string = 'This is a very long string'
if 'long' in long_string:
    print 'Match found'
```

Listing 1: An example of Python code

3.4.2 Naming

As well as the readability which comes with Python, further efforts will be made to ensure all classes, methods, and variables will be named as descriptively as possible. This makes the system and its algorithms easier for an outsider to understand should if any parts of the code need to be revisited at a later date. This is demonstrated in Listing 2, where although the left-hand code looks cleaner, when read through looks extremely generic and could be part of any system. In contrast the right-hand code is much more descriptive of its function.

```
public List<int[]> getThem() {
  List<int[]> list1 = new
   ↪  ArrayList<int[]>();
  for (int[] x : theList)
    if (x[0] == 4)
      list1.add(x);
  return list1;
}
```

```
public List<int[]> getFlaggedCells() {
  List<int[]> flaggedCells = new
   ↪  ArrayList<int[]>();
  for (int[] cell : gameBoard)
    if (cell[STATUS_VALUE] == FLAGGED)
      flaggedCells.add(cell);
  return flaggedCells;
}
```

Listing 2: Examples of bad naming (left) and proper naming (right) taken from Clean Code (Martin, 2009)

3.4.3 Object Orientation

Object orientation is the use of classes which contain their specific variables and methods to protect functionality from other parts of a system. An object oriented approach will be taken when creating the system to ensure its functions and data is protected should it ever be used as a module elsewhere. The use of private variables and methods with one or two public methods provides an Application Programming Interface

(API) with which other developers can use the system.

Python does not provide the private and public keywords as found in the Java or C++ languages. It instead identifies methods and variables prefixed by one or two underscores as protected. This means these items will be hidden from view when the system is used but remain accessible should the developer require them. The creators of Python chose this method as they enforce responsibility over restriction.

3.5 Testing

To verify the functionality and quality of a software application it must always be tested. This can be carried out at a variety of levels from testing a single class method to an entire system and by knowing how an application works in white-box testing or being unaware of the functionality in black-box testing.

3.5.1 Unit Testing

Unit testing is carried out on individual units of source code to ensure they are functioning correctly. Normally carried out in the scope of an individual class or module, these unit tests are typically written by the same software developer who produced the class itself. A single unit test comprises of a set-up process where data and objects are initialised, the test itself including an assertion on a variable determining pass or fail, and a tear-down process where any changes the test has made are cleared up.

As unit tests are simple in structure and quick to run they are commonly executed when changes are made to the source code. This verifies that the changes made have not broken other sections of code and can be committed into the master branch successfully. If a test failure does occur then each unit test should be concise and descriptive enough to aid the debugging process by indicating what has failed and where.

A metric of quality for unit testing is code coverage. To confirm that a class is of good quality and functional, every possible path through the code must be tested. Testers should aim for 100% coverage of a module meaning every possible piece of functionality has an associated test with a coverage of 85% for example meaning that 15% of a module is not under test and could cause undetected problems if any changes are made. Many Integrated Developers Environment (IDE) or testing suites provide coverage checking functionality to make this process simple.

3.5.1.1 Python Unit Testing Due to the package based and open-sourced nature of the Python programming languages there are many implementations of unit testing frameworks to choose from. Each has its own advantages and disadvantages and there is much debate over which framework is the best to use. A good unit testing framework should provide the necessary functions to create simple unit tests for each path in the source code and be substantial enough to ensure the class or module is correctly tested.

3.5.2 Project Testing Scope

Due to the small size of the application which will be produced in this project, testing will be limited to unit testing as there is no integration to be carried out. This small size also means that 100% code coverage will be simple to achieve. From this, stability and functionality of the application can be verified improving the quality of the overall product.

Once the basic functionality has been added to the application, unit tests will be created covering all source code produced including normal operation and sections which can potentially produce errors. This means any subsequent changes can be tested to see if they have affected functionality. Any new functionality will have tests added to the testing suite to maintain 100% coverage.

The Python unit testing framework chosen is unittest2. Although it has been superseded by Python3, Unittest is the default framework used by the PyCharm IDE. The unittest2 module provides a backport of this functionality for use in Python2 meaning extra testing functions are provided to allow more suitable tests to be created. PyCharm also provides the ability to run tests with code coverage producing a clear indication of which sections of source code are covered and which are not; this will be greatly beneficial to achieving full coverage.

3.6 Project Management

3.6.1 Agile Development

To do (9) Introduced in 2001 with the writing of the Agile manifesto (Beck et al., 2001), agile development methodologies focus on high quality software products over the rigorous design-based structure of traditional waterfall methods. By utilising various techniques such as stand-up meetings, a strong customer focus, and sprint development cycles, agile has become widely adopted in industry and has been proven to produce successful projects that are both more closely tailored to the customers' needs in shorter time scales (VersionOne, 2015).

There are numerous agile methodologies (XP, Scrum, DSDM) each with their own principles and techniques. However it is commonplace for a company to create their own agile approach picking and choosing items from different methodologies to suit their particular needs (Aydin, Harmsen, Van Slooten, & STEGWEE, 2004). As this is a solo project with no customer then a single set methodology will not be used. Instead a variety of methods will be used to help manage the project and drive the efficiency of the development process. These will be outlined in the following sections.

3.6.1.1 Requirements Analysis Used in some form by the majority of agile methodologies (Cao & Ramesh, 2008), requirements analysis or requirements engineering is a variety of processes used to create the conditions that a project must meet. These requirements take into account stakeholders, users, and the development team or teams. Each requirement should be documented, actionable, measurable, testable, and traceable to aid in its understanding and completion.

The techniques used to produce requirements include stakeholder identification and interviews which are used to identify what the stakeholders of the project require once it is completed. These interviews may be followed by Joint Requirements Development Sessions which bring stakeholders together to further discuss the requirements of the project. A more traditional approach is to produce a contract-style requirements list although these are typically extensive and incomplete as considerable collaboration is involved in producing them. The requirements analysis techniques that are most frequently used in agile developments are use cases and user stories. These are either written or diagrammatic descriptions of how the system will interact with users or other systems and provide an indication of what will be required from the product.

Due to the prototype nature of the application being produced and as the project has no defined stakeholders, the only requirements analysis technique which will be used is production of use cases. These will aid in identifying how the application will be operated by users and what the project will need to produce. These requirements will then be prioritised using the MoSCoW format and inserted into a tracking system for the development process as described in the following sections.

3.6.1.2 MoSCoW First used in the DSDM agile framework (Bittner, 2002), MoSCoW analysis or prioritisation is the process of taking the requirements of a software product and placing them into one of four categories; Must, Should, Could, and Won't have. These deliverables may then be prioritised either for the entire project or for individual sprint cycles depending on the size and magnitude of the project.

MoSCoW was created to provide customers a better understanding of software requirements. Unlike categorizing priorities as high, medium and low, MoSCoW is more descriptive of what the prioritisation means for the software project. From a development point of view, this prioritisation process allows developers to focus on the core requirements of the project first, creating a viable software solution early in the development cycle with less important features being added later if the resources are available.

This project **To do** (10) will use MoSCoW to prioritise the requirements identified using the requirements analysis technique allowing the development process to complete the goals in the correct order. This will ensure that the core aspects of the solution are implemented first creating a successful project early and allowing it to improve as time allows.

3.6.1.3 Sprint Cycles **To do** (11) Used by Scrum development teams (Rising & Janoff, 2000), a sprint is a timeboxed effort of work scheduled to take between one week and one month. At the start of a sprint, goals are chosen from the project requirements which are to be completed by the end of the sprint. When a sprint is complete a retrospective review is carried out by the development team to discuss what went well, what didn't, and how this can be rectified in the next sprint.

This project will use sprints in the same manner as an agile development team as this will allow easier completion of requirements and tighter management of the development process. Using sprints will ensure the time allotted for development is used effectively which will lead to a higher quality product at the end of the project.

As mentioned previously, MoSCoW prioritised requirements will be selected at the start of each sprint cycle to set objectives the work which will be carried out. Each cycle will aim to complete all of the must have requirements and the majority of should and could haves. At the end of each cycle, a retrospective review will be carried out which will re-prioritise any uncompleted requirements based on how successful the sprint cycle was. This means requirements may be promoted or demoted respectively.

3.6.2 Organic Development

An alternative method to agile development could be to take a more organic approach to the development process. This method would take the step-by-step ethic of agile and reduce the effort applied for analysis, documentation, and structuring of the working schedule.

To initiate the development process without analysis of requirements would be to begin producing the basic steps or functionality which the system will go through from a vision of the end product. Once started, development would take a natural direction identified by what is needed for the system or to solve any issues encountered.

To document the development process, a daily log of any work completed would be kept whenever any development is carried out. This would allow for referral at later dates and keep the project on track by identifying when features are completed.

There are advantages and disadvantages to using this method over an agile process. Due to the size of the software being produced the amount of management work inherent in an agile process could mean the development process is over managed

3.6.3 Version Control

Large software projects produce multiple files of code, data and documentation. These are are vital to the overall success of a project and should be kept safe. Were these files to be lost then the project could easily be delayed or drawn to a close as the time and resources may not be available to recreate the lost data. To combat this any data relating to the project must be backed up, preferably on a cloud based system, to avoid loss and allow the project to continue should anything happen to the local copy of the data.

For this project the Git Version Control System (VCS) will be used. Git uses a cloud hosted repository to store any files relating to a project and allows for work to be carried out locally by cloning the repository on a computer. However VCSs also enables the management of the previous versions of files including information such as the individual changes made, details of when those changes were made, and who made them This is a powerful tool as it means the various sections of the project can be worked on without the risk of damaging the project and should a change prove unsuccessful then the repository can be reverted to a functional point.

For this project the web service used to host the repository will be github.com. This site was chosen as it provides unlimited free repositories as well as simple repository management tools. The website also provides issue tracking functionality which will be used to log each new feature as it is added will be added, although strictly speaking they are not issues at all. The reason for this is that all features logged in the repository will have a unique identification and description against which commits to the recorded. This is useful for project management reasons as then each feature's stage and completeness can be monitored to ensure the project's success.

Alternative VCSs, such as Microsoft's Team Foundation Server or Perforce, are available. However these are limited under free licences or locked to certain development Environments whereas the accessibility and ease of use provided by Git makes it the optimal VCS to use for this project.

3.6.4 Milestones

Milestones are used in project management to mark significant events or points within a project's timeline. This allows a further breakdown of the project into milestones which can then be used as intermediary deadlines. The use of milestones allows the project manager, management team, or development team to keep track of the project's status and priorities at any given time.

This project has, and will use, milestones for exactly the same reason. Current examples include the week nine review session and the deadline for the completion of this document. As more milestones are identified through breaking down the development process into development sprints and other ancillary tasks, they too will be documented and acted upon.

3.6.5 Threshold

Tasks taking up more than their allotted time is a common cause of projects consuming more resource than initially allocated. To ensure all tasks can be completed within the allotted time for this project, each task will be allocated a threshold. Adding a threshold is a common technique in project management and provides extra time should anything unexpected occur which impacts the timely delivery of the project.

3.6.6 Gantt Chart

A Gantt chart is a method of breaking down a project into tasks and graphically representing them as bars on a chart first used by Henry Gantt in the second decade of the 20th century as a way of ensuring schedules are met. Commonly starting at a project's inception date and ending with it's completion date, tasks are allocated an estimated duration for their completion and placed within the timeline. These tasks can then be allocated dependencies to indicate their prerequisites and graphically show the project's critical path.

This project will make full use of the Gantt chart technique by decomposing the project into multiple stages for both writing and development. Previously mentioned milestones will also be added for the known fixed points. Each sprint cycle for the development process will be indicated with further notes on the tasks to be carried out. Microsoft Project 2016 will be used to create the Gantt chart.

3.7 Evaluation

To ensure that the application produced is extensively evaluated, multiple methods will be used. Each of the different methods will evaluate a single aspect of the application from varying points of view.

3.7.1 Validation

This is a measure of how well a design or solution meets its original specifications and fulfils its intended purpose. It involves checking that all required functionality is present, each output is correct and as expected, and the solution performs as expected.

In this project, validation will be provided through the implemented unit tests. Ensuring the application has near 100% coverage and that it has successfully passed each test will confirm that the application is operating as expected.

3.7.2 Reliability and Accuracy

Assessing the reliability and accuracy of the application proves that the results that it produces are both consistent and correct. To assess these an uncertainty metric will be produced as described below. When taking multiple measurements there is typically a ”true” value which is the actual measurement that falls somewhere within the range of produced measurements. Uncertainty is a prediction of how close any produced measurement will be to this ”true” value, expressed as $\overline{x} \pm U$. The method for producing a value for uncertainty is as follows:

1. Produce a suitable number of repeat measurements $\{x_0 ... x_n\}$
2. Calculate the average value of these measurements $\overline{x} = \frac{\sum\{x_0 ... x_n\}}{n}$
3. Find the differences between the measurements and the average $\{d_0 ... d_n\} = \{(x_0 - \overline{x}) ... (x_n - \overline{x})\}$
4. Calculate the average of these differences squared $\overline{d_s} = \frac{\sum\{{d_0}^2 ... {d_n}^2\}}{n}$
5. Produce the uncertainty or standard deviation of these results which is the square root of the average differences $U = \sqrt{\overline{d_s}}$

This process will be carried out a number of times altering all possible variables depending on what the application is capable of once produced. These variances will be outlined once the evaluation has been completed.

3.7.3 Comparison to Alternatives

Comparing the process of using the application to produce a sag setting against other available methods will show how simple the application is to use and whether it is more effective than other methods. For this comparison, outputs from the application will be compared with a trial and error process that is a common practice for a beginner or intermediate rider setting their suspension for the first time as described in section 2.1.4. If the manufacturer's recommended setting prove to be incorrect, then pressure is increased or removed by set increments. Comparison to this process will indicate whether the application presents a benefit over the manual approach.

3.7.4 Professional Opinion

The final method of evaluation will be to seek the opinion of the application from professionals working within the mountain bike and cycling industry. This will provide a further indication of how well the application achieves its goal and how it might be received were it to commercialized as well as suggesting areas for future work which may not have been seen without outside consultation.

The Mountain Bike Research Centre of Scotland[2] will be approached to recommend an individual who they deem suitable to provide such expert assessment once the application can be demonstrated. Although it is anticipated that the meeting will take a semi-formal approach, general evaluative questions will be posed with responses recorded on an appropriate scoring scale where appropriate.

In addition to this meeting, the application will also be demonstrated to staff at a local bike shop to also gain their feedback using the same approach as the meeting with the individual designated by The Mountain Bike Research Centre.

[2] http://www.napier.ac.uk/about-us/our-schools/school-of-applied-sciences/research/mountain-bike-centre-of-scotland

4 Results

This section will cover how the project was created and evaluated. It will begin by describing any deviations from the planned development process including any problems which were encountered and how they were addressed. It will then continue to describe how the final version of the application operates with a pseudocode description. Finally it will present the previously outlined critical evaluation covering validation, reliability, alternative comparison, and professional opinion.

4.1 Deviation from Plan

4.1.1 Development Approach

As described in section 3.6 there was some uncertainty about which approach would be followed during the development stages of the project. As it happened, once the development stage was reached, an organic approach was taken. Although less structured than an agile methodology it was decided that removal of the sprint process would allow development of the application to progress quicker as items would not need to be delayed for the next sprint cycle. Allowing problems to be addressed immediately proved useful as the application was not left in an unusable state while other features were produced.

The log produced by the development stage can be seen in Appendix C. This shows that an adequate amount of work was carried out within the time-frame allocated despite using a less structured process and that the final application includes complete functionality to meet its stated aims.

4.1.2 Use of EXIF Data and Reference Point

As mentioned in section 2.2.4.4, the process which the application would use to produce measurements was undecided and required experimentation. Initially Exchangeable Image File Format (EXIF) data from the image was used, which was successful until collection of data about the sensor size was required. Due to this being uncommon in EXIF data, particularly in images taken with smartphone cameras, this method was abandoned in favour of using a reference point. The code produced during this experimentation phase can be seen in Appendix D where the function `get_sensor_size` is incomplete as this is the point where this approach was dropped.

The reference point chosen is a red circular sticker as these are cheap, freely available in store or online and being round do not need to be orientated to the object as a square or rectangular sticker would. This sticker is applied directly to the shock unit where it can be easily picked out in the image due to its prominent by its colour. But although locating the reference point was simple, issues were encountered in the measuring process and these will be discussed in a following section.

4.1.3 Colour Quantification

For initial experimentation with using a reference point, a similar sized red circle was manually added to images using an image processing application. This allowed for tuning of the colour masking process and refining the process before using a real reference point stuck on the shock. However when images with a real reference point were used, the application could not detect it. This was because the colour range for the masking process was using perfect red (RGB 255,0,0) and the non-standard reds of the sticker as captured in the image were not within this range.

A first attempt to resolve this issue involved expanding the boundaries of acceptable colours that the application would recognize in the image as being perfect red. This proved to be insufficient so a further image processing technique known as colour quantification was also used. This reduces an image to 2, 4, 8, 16, or 32 bit colour spaces which sacrifices the range of colours contained in an image detail but makes objects more prominent so they are easier to detect. The difference between an original image and one using quantified colours is shown in Figure 14. Quantifying the colours to 8 bit makes the reference point a flat tone of red while maintaining its shape allowing for the masking process to function correctly.

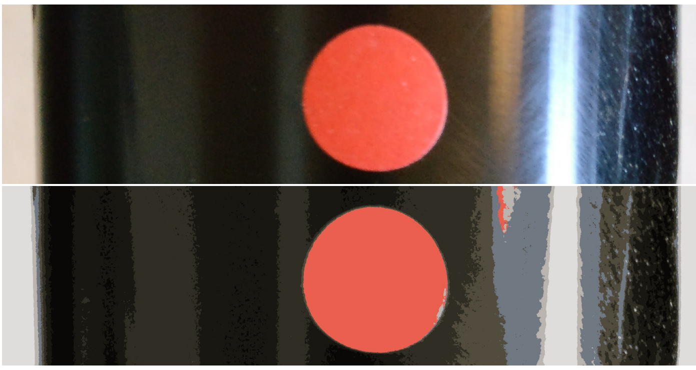


Figure 14: Normal image (top) versus quantified colours (bottom)

4.1.4 Dynamic Measurement Limits

It was decided that the application should utilise the distance between the shock wiper seal and the marker o-ring for providing the measurement. This was deemed suitable as the manual process to calculate sag also uses the O-ring which is a prominent feature on most shocks, unless removed by the owner. Initial versions of the application assumed it would find the O-ring somewhere about two thirds of the way up the image height although once multiple images were used from two different makes of shocks this assumption proved to be incorrect.

To address this and ensure the application is always able to find the O-ring no matter where it appears in the image, it was necessary to use a dynamic method to detect it. In much the same way that using quantified colours solved the issues of detecting the reference point, OpenCV's findContours can be used to find an O-ring of a known colour as long as it is present in the image. When located, a bounding box is drawn around the contour as shown in Figure 15 and this can be used to determine the lowest point at which the O-ring appears in the image to establish the limit of measurement.

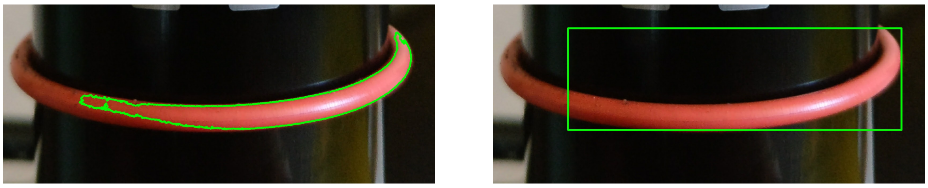


Figure 15: Red O-ring found using `findContours` (left) with `boundingBox` applied (right)

As some shocks do not have a coloured O-ring, this process was adapted to locate a black O-ring. This was not successful and an alternative method was used where thresholding is applied to the image and contours produced from the reflections on the telescopic shaft of the shock unit. This made it easy for the application to detect the discontinuity between the O-ring and the shaft so the same bounding box could be applied and used to establish the lower limit for where the application can measure. This process is shown in Figure 16.

Figure 16: Black o-ring found

4.1.5 Reference Point Measurement Method

To measure the dimensions of the circular reference point, the first process applied was Hough Circle Transform as described in 2.2.4.3. This method was chosen as it is designed specifically to find circles within an image and is simple to implement. Although this did produce measurements, once these values were compared with values derived by manually measuring the image, they proved to be unreliable with variances both above and below the actual value. This effect is shown in Figure 17 where it is clear that the circle identified using Hough Circle Transform is much smaller than the actual reference point.

To overcome this issue it was necessary to again use OpenCV's findContours in exactly the same way it used to successfully locate the O-ring. Once a contour is found that encompasses the entire reference point, a bounding circle can be drawn around it as shown in Figure 18. Although the bounding circle appears to be slightly larger than the reference point, experimentation with a variety of images showed that this method

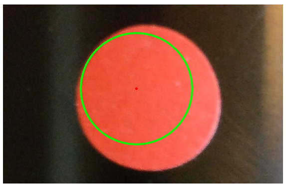


Figure 17: Reference point found using HoughCircles

produced measurements that were within acceptable tolerances for the pixel per millimetre calculation.

The reason the Hough Circle method produced a bounding circle that is slightly difference to the reference point is because the reference point captured in the image is not a uniform circle. Although the thresholds of the Hough Circle method can be adjusted, applying the reference point to the cylindrical shaft of the shock unit means it is too misshapen to be detected correctly without the use of OpenCV's findContours.

Figure 18: Reference point found using findContours (left) with boundingCircle applied (right)

4.1.6 Pressure Calculation

The original method for producing a suggested pressure that would give the optimal amount of sag was to calculate a psi per millimetre metric from the measurement of the shock shaft and the current pressure of the shock supplied by the user. The assumption being that for every 1 PSI added to the shock, this would reduce the movement by a fixed amount proportionate to the original pressure. The equation for this metric is as follows:

$$P_s = \left(\frac{P_c}{M_c}\right) M_s \tag{2}$$

where

P_s is the pressure to produce the desired sag setting
P_c is the pressure currently in the shock
M_s is the measurement to produce the desired sag setting
M_c is the current measurement of the shock shaft

Though this hypothesised method had produced reliable results when tested manually, if failed to do so in the application both when different images, reference points and desired sag value were test and even when these values were hard-coded. Further investigation and discussion suggested the cause of this problem was the non-linearity of air springs as previously discussed in 2.1.3.

When an air spring is compressed the spring rate increases through its travel as the particles of air become increasingly compressed (Goodyear, 2014); While this effect still allows the shock unit to be fine tuned, applying a linear equation to a non-linear phenomenon is clearly going to produce some degree of error. However as the range of measurement in this project is small compared to the full stroke of an air shock, it was considered that the spring rate can be considered as linear within the limited range of compression under consideration.

To investigate this, the sag of a shock was measured at pressures between 100 and 250 psi in increments of 10 psi. This was carried out for two shocks, one with a normal air can and another with a high volume air can, then the collected data was plotted and compared against a trend line generated by linear regression. The results of this can be seen in Figure 19.

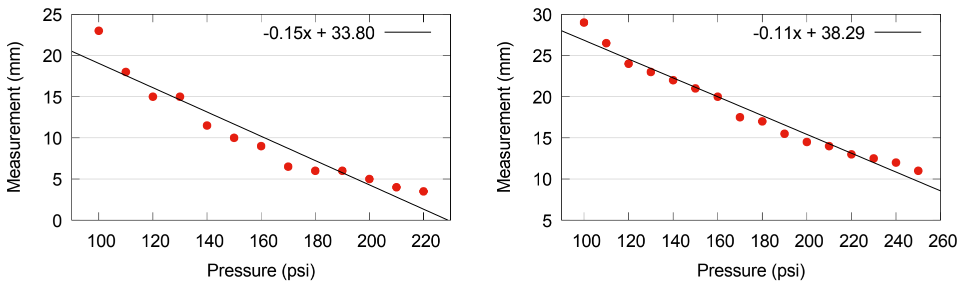


Figure 19: Sag measurements for standard air shock (left) and high volume air shock (right)

From this data it could be concluded that, within the range of measurements, the compression of the air spring linear. The data-points in Figure 19 do occasionally vary though and this thought to be due to it being a non-sterile test environment. Both shocks were well-used potentially containing age-related defects and loading the bike was done simply by using body weight.

Having established that the relationship between the pressure in the shock and movement from weighting the shock could be treated as if it were linear, the plan was to implement an ideal gas equation. However investigation into producing charts using Python uncovered a method of computationally producing a linear equation. This led to the application's process being rearranged so that it takes in two images, one of the shock loaded at 100psi and another at 150psi. Using this data it can produce a virtual plot and linear equation with which to calculate a value for the optimal sag setting.

4.2 Application

4.2.1 Images

The application requires two images to measure and produce calculations from. The application can process images in different lighting conditions and ones where a flash has been used though there are some criteria which are needed. There must be a red, circular, 7.5mm diameter sticker on the shock, preferably just above the shock seal. The image should also be taken so that the shock seal is around 1/3 down in the image.

Examples of the images used are shown in Figure 20.

The user must supply one image of the shock which has been pressurised to 100psi and a second with the shock pressurised to 150psi both loaded under the normal riding weight, i.e. complete with helmet, body armour and hydration pack. Each time the marker O-ring should be returned to the top of the shock shaft before weight is applied to the shock.

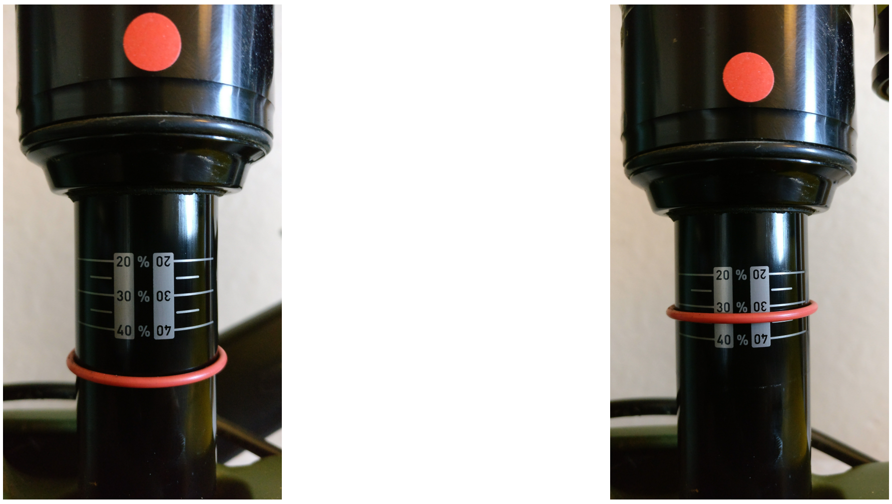


Figure 20: Images suitable for processing

4.2.2 Arguments

The application operates from a command line interface allowing the user to supply data through arguments; these are described in Table 3. This system uses Python's Argparse package and a full command to run the application would be as follows: `program -i user/images/100.jpg user/images/150.jpg -s 30 -t 57 -c red`

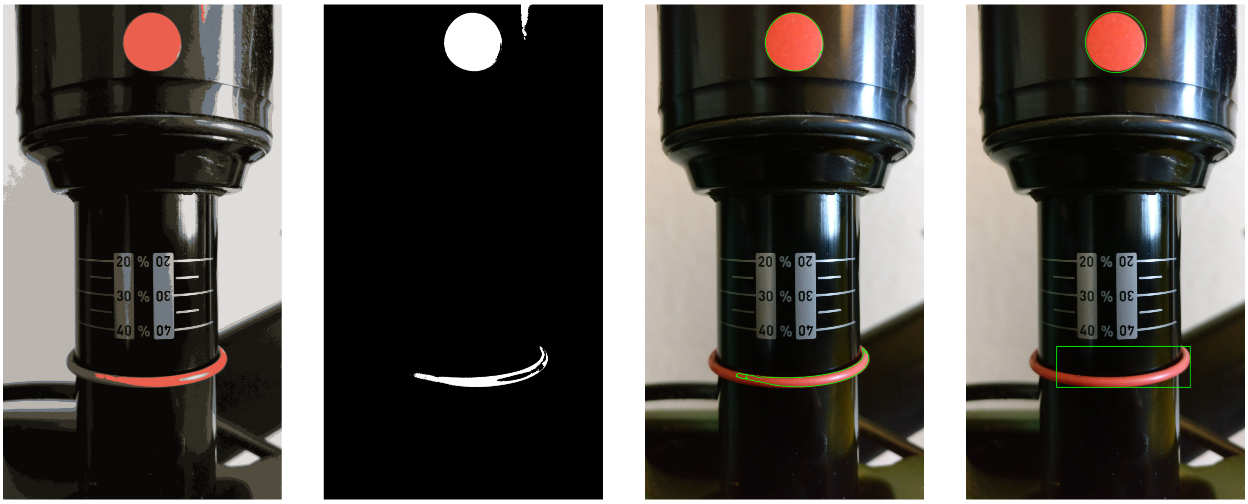
Table 3: Table of application arguments

Argument	Description	Usage
-i,--image <path1 path2>	The paths to the two images to be used for analysis	-i path/to/img/1 path/to/img/2
-s,--sag <percentage>	The desired sag percentage	-s 30
-t,--stroke	The stroke length of the shock being analysed	-t 57
-c,--colour	The colour of the marker o-ring	-c red
-d,--debug	This produces outputs from each stage to the terminal for debugging purposes	

4.2.3 Processing

4.2.3.1 Reference Point The first process which the application must complete is to find the reference point for measurement. This is done by first colour quantifying the image to 8 bit colour so that the reference point becomes uniform in colour. The result of this process is shown in Figure 21a. So the application can find the red colour of the reference point within the entire image, a mask is applied for any colour within a range. The output of this process is shown in Figure 21b. If an alternative colour reference point were to be used, this colour range could be easily adapted.

The contours,or shapes in this image are then found and sorted by their area in decreasing order. By doing this it can be assumed with a high level of confidence that the largest will be the reference point. The found contour can be seen in Figure 21c. A bounding circle is created around the reference point contour, as is shown in Figure 21d, and this can be used to calculate the pixel per millimetre metric.



(a) Colour quantified image

(b) Image mask

(c) Found contours from mask

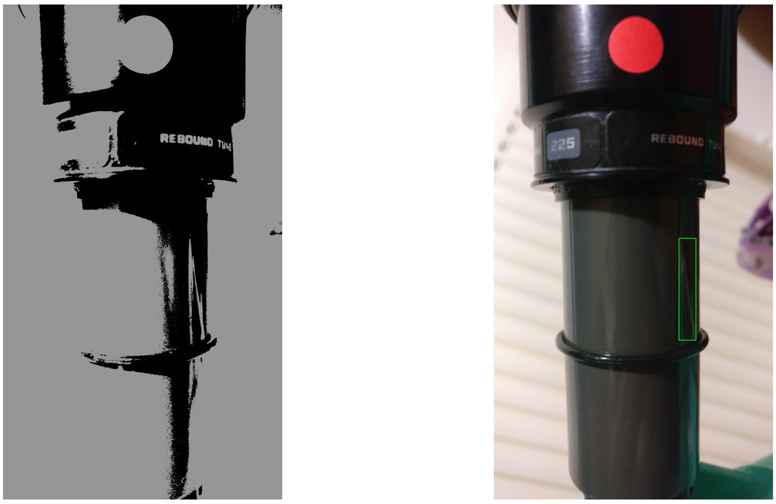
(d) Bounding circle and box from contours

Figure 21: Processed images for finding reference point

4.2.3.2 O-Ring If a red coloured O-ring is specified then the same process to find the reference point is used. The second largest contour found will be the O-ring so a bounding box can be drawn round it from which a measurement limit can be collected. The red O-ring mask, contour, and bounding box can be seen in Figure 21.

If a black O-ring is specified, an alternative process is used. Thresholding is applied to the image which highlights any shadows or highlights on the shock shaft. As these are intersected by the O-ring it is simple to select a contour which ends at the O-ring to use as a measurement limit. This process is shown in Figure 22.

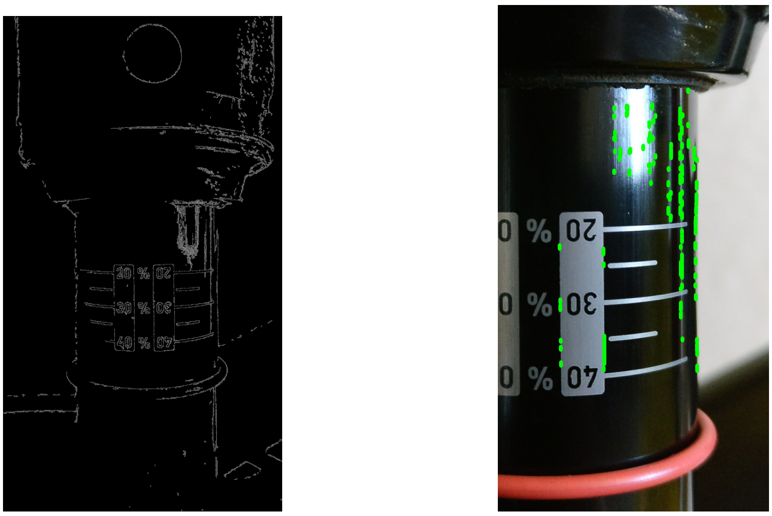
4.2.3.3 Measurement With the pixels per millimetre metric produced and the measurement limit found, the sag under the current pressure is measured. First, edge



(a) Image thresholding (b) O-ring indicator

Figure 22: Locating process for black o-ring

detection is carried out on the original input image to highlight the lines of the shock, as shown in Figure 23a. Hough line transformation is then applied to locate vertical lines between the top of the shock shaft and the measurement limit. The difference in pixel count between the highest and lowest points of these lines represents the current sag measurement in the image and this can be converted to millimetres using the px/mm metric.



(a) Edge detected image (b) Vertical lines in measurement area

4.2.3.4 Equation Once the measurements for the shock pressurised to 100psi and 150psi are produced, they can then be utilised to create a linear equation. This is carried out using Python's Scipy package and its linregress method which accepts two arrays of data and returns the slope and intercept of the linear equation. An example equation which is visually described in the chart shown in Figure 24 is only produced virtually and never output to the user.

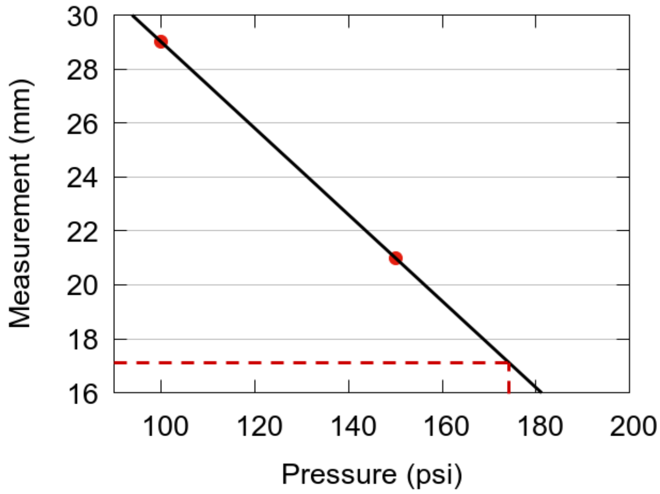


Figure 24: Example of linear equation plot

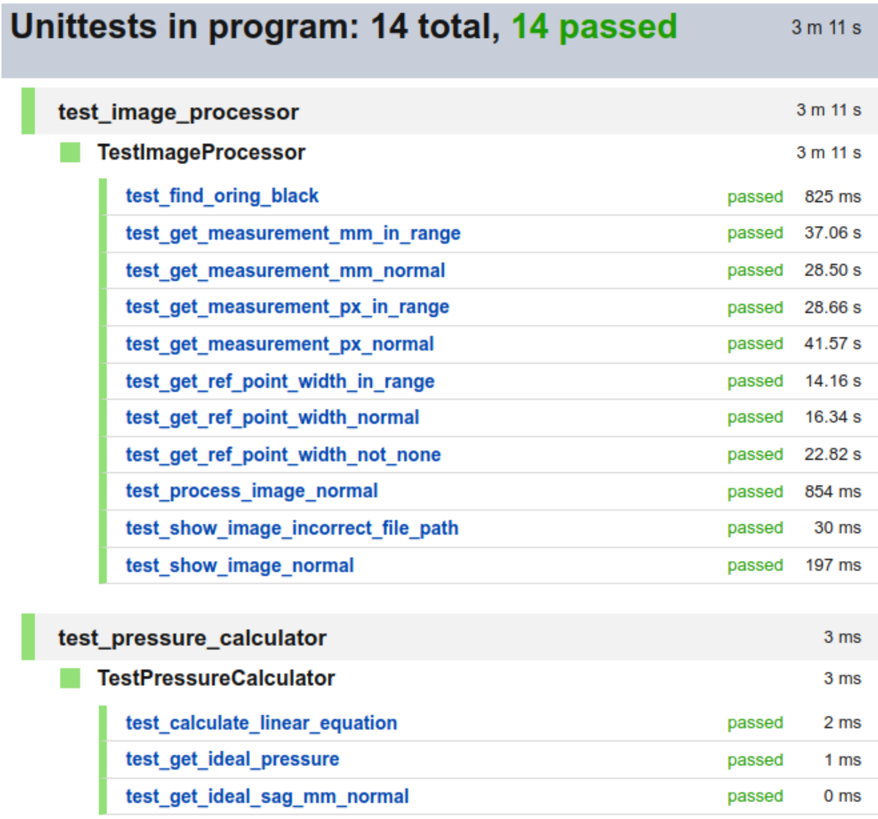
4.2.3.5 Produce Setting To produce the final setting the optimal distance that the shock should move to give the desired sag setting is first calculated using data input by the user. For example if the shock stroke is 57mm and the desired sag is 30% then the desired movement is 17.1mm. This and the equation produced in the previous are used to calculate the optimal pressure. **To do** (12) For example if the equation is $-5:017x + 246.426$ then the pressure required to give optimal sag is 161psi.

4.3 Evaluation

The following section will describe the results of applying the evaluation techniques outlined in section 3.7 to the final prototype application. The outcome of the evaluation presents an indication of how successful the application is at its intended goals and how successful the project was in producing it.

4.3.1 Validation

Figure 25 shows an extract from the testing report produced by the Pycharm IDE. It includes each test from both test modules and clearly indicates that the final prototype application successfully passed all the tests which included checks to see if the outputs produced by the application are within an expected range, validating not only that the application works but that it also produces results within the anticipated range.



Unittests in program: 14 total, 14 passed 3 m 11 s

Test	Result	Time
test_image_processor		3 m 11 s
TestImageProcessor		3 m 11 s
test_find_oring_black	passed	825 ms
test_get_measurement_mm_in_range	passed	37.06 s
test_get_measurement_mm_normal	passed	28.50 s
test_get_measurement_px_in_range	passed	28.66 s
test_get_measurement_px_normal	passed	41.57 s
test_get_ref_point_width_in_range	passed	14.16 s
test_get_ref_point_width_normal	passed	16.34 s
test_get_ref_point_width_not_none	passed	22.82 s
test_process_image_normal	passed	854 ms
test_show_image_incorrect_file_path	passed	30 ms
test_show_image_normal	passed	197 ms
test_pressure_calculator		3 ms
TestPressureCalculator		3 ms
test_calculate_linear_equation	passed	2 ms
test_get_ideal_pressure	passed	1 ms
test_get_ideal_sag_mm_normal	passed	0 ms

Figure 25: Unit test results

Figure 26 shows the test coverage report for the application. As stated in section 3.5.2 a goal for this project was 100% test coverage of the application. This has not been achieved but has come close with 97% of the application being covered.

The main.py file has no coverage as it is a script as opposed to a testable Class. This is not an issue as it contains the argument parsing and calls to other classes which are not vital to test as the Argparse module is well tested already and the lower level classes have their own testing. The image_processor.py class does not have full coverage as this includes lines which check if outputs provided by OpenCV are valid and close the application if something has gone wrong. The conditions under which these lines are hit could not be recreated due to the irregularity with which problems occur. However testing these lines would not validate the application's normal functionality.

83% files, 97% lines covered

Element	Statistics, %
image_processor.py	93% lines covered
main.py	not covered
pressure_calculator.py	100% lines covered
test_image_processor.py	100% lines covered
test_pressure_calculator.py	100% lines covered
utils.py	100% lines covered

Figure 26: Test coverage results

4.3.2 Reliability and Accuracy

Table 4 shows the results of calculating uncertainty for the application under different conditions. The calculations were carried out for two different shocks with two different coloured O-rings, and for two sag settings on each. For the tests the application was run 10 times using the same images and arguments with every output collected, and used in the calculation.

Table 4: Table of uncertainty calculation results

	High Volume, Red O-Ring		**Low Volume, Black O-Ring**	
	25%	**30%**	**25%**	**30%**
Measurements (PSI)	175.02	160.97	133.31	120.54
	174.84	160.52	133.28	120.47
	174.95	160.41	133.42	120.48
	174.83	160.65	133.42	120.46
	174.94	160.75	133.29	120.53
	174.86	160.50	133.38	120.51
	175.63	160.48	133.35	120.50
	174.78	160.77	133.36	120.46
	175.29	160.57	133.36	120.46
	174.84	160.74	133.42	120.50
Average	175.00	160.64	133.36	120.49
Standard Deviation	0.27	0.17	0.05	0.03
Uncertainty	175 ± 0.27	160.64 ± 0.17	133.36 ± 0.05	120.49 ± 0.03

For each test the results are low, (the highest being ± 0.27), which indicates that the application is capable of consistently producing correct sag settings. If these uncertainty measures had been much higher than the settings produced by the application would not have been reliable. The calculated setting produced by the application is also rounded to the nearest whole number as increments of less than 1 PSI in a mountain bike shock have negligible effect. Even the most advanced, digital shock pumps do not display fractions of a PSI and professional race teams do not make changes of less than 1 PSI. This knowledge further reinforces the capabilities of the application.

The results for the shock with a red O-ring are slightly higher than those for the black O-ring. This is because the method for finding and applying a boundary to the red O-ring produces a larger variance in results than the method for finding and applying a boundary to s the black O-ring.. However, the minor difference between the results is insignificant as both colours of O-ring produce accurate results so this is not an issue.

4.3.3 Comparison to Alternative Methods

As outlined in section 3.7.3, the results generated by the application were compared with the manual, trial and error method of setting up sag. For this experiment the target sag was 30% of a 57mm shock stroke equalling a target measurement of 17.1mm. To begin the rider wearing their normal riding clothing and equipment were weighed with the result being approximately 160lbs. Following the manufacturer's recommended process, the shock was then pressurised to the equivalent pressure of 160 PSI and loaded with the rider in the saddle with the feet off the ground. Each of the subsequent steps followed in the trial and error process that are shown in Table 5.

Table 5: Stages of manual sag setting process

Stage	Pressure(PSI)	Outcome	Action
1	160	Too little	+5
2	165	Too little	+20
3	185	Too little	+5
4	190	Too little	+10
5	200	Hit target	N/A

To produce the correct sag setting using the manual process required 5 adjustments each including pressurisation of the shock, weighting of the bike, and measurement of the marker O-ring. For this particular shock, small changes in pressure made no noticeable difference to the amount of movement so and additional 20 PSI was added in stage 2. Even though there is a chance that the manufacturer's recommended setting will be correct or very close, the majority of the time a manual set-up needs 4 or more adjustments. Conversely, using the application only involves 3 set easy to follow stages and produced reliable results that meant the optimal setup can be achieve quicker and with far less effort. The application also removes both the need to weigh the rider and to refer to the manufacturer's recommended pressure setting which further simplifies the process and saves time.

It should be noted that there is disparity between the resulting pressure of the manual process (seen in Table 5) and that produced by the application (seen in Table 4). As the images used during development of the application were of a shock that had not been weighted while the rider was wearing correct clothing and equipment, new images were produced to match the conditions during the manual process and the application retested. This produced a result of 176 PSI closer to that produced by the manual process but still lower than the manually produced result. However, when the shock is pressurised to this setting, the sag appears correct.

A manual setting was produced for the low volume shock to further investigate this issue. It was found that under these conditions both the manual and application's results were extremely close at approximately 120 PSI. As high volume shocks require much higher pressures than low volume shocks, it is believed that as the manual pressure of 200 PSI lies above the two pressures which the shock is pressurised to when using the application, this creates an inaccuracy in the results. For comparison, the low volume target pressure of 120 PSI lies between the 100 and 150 PSI required by the application. A solution to this will be discussed in section 5.3.

4.3.4 Professional Opinion

To do (13) As outlined in section 3.7.4, once the application was complete the Mountain Bike Centre of Scotland were contacted to arrange a meeting with an individual from the industry to demonstrate the application to and collect their feedback. The meeting was arranged with Geraint Florida-Jones of the Centre and Mark **To do** (14) who is a mechanical engineer who has ventured into the design and manufacture of his own full suspension frame. Both Mark and Geraint are familiar with the current products that are similar to the application produced by this project such as Shockwiz (Dusty Dynamics, 2016) and the Fox IRD application (Benedict, 2012).

Once Mark and Geraint were familiar with the context of the project and operation of the application, they were asked for their feedback. They both stated that the project had been well executed and the application was simple to operate. Both verified that the setting produced appeared correct for the given shock, bike, and rider's weight. Furthermore, both agreed that they would recommend the application to the target audience of beginner and intermediate riders though a more professional setup was required for those at the competitive end of the sport.

Both said that the application produced in this project should be released as a mobile application as this would make it much more user friendly compared to the prototype command line application. Additionally, both stated that more features should be added to provide a more rounded user experience and their suggestions will be explored in section 5.3. Mark questioned the application of a linear equation to a non-linear air shock though once the experiments into linearity, outlined in section 4.1.6, were explained he agreed that the shock could be considered linear within the range of its stroke that sag can be adjusted.

The application was also demonstrated to the staff at Pedals[3] bike shop in Edinburgh. Here it received much the same feedback as ch it got during the meeting with Mark and Geraint; namely that it is a good execution of the concept and with more features would make a mobile application that is entirely suitable for the target demographic.

The positive reception that the application received from professionals within the cycling industry demonstrates that it is a suitable prototype which produces correct settings in an easy to use manner. The only issues suggested were the lack of additional features and functionality which is addressed in section 5.3.

[3] http://pedalsbikecare.co.uk/

5 Conclusions

This section will conclude the project by evaluating how well the original aims have been met to determine the success of the project, compare the prototype application to current products on the market, and outline any future work which can be potentially completed in subsequent projects. Finally a self appraisal will be carried out to discuss this author's performance during the project.

5.1 Meeting Aims

As a measure of success the original project aims which, were explained in section 1.3, can be examined to see if they have been met upon project completion.

5.1.1 Aim 1

The first aim was to complete a literature review of mountain bike suspension and image analysis techniques including how image analysis is currently used in sports science which has been met by section 2. This literature review presented the fundamentals of mountain bike suspension to provide insight into its operation as a way of building knowledge of the project context. The research into image analysis uses and techniques proved useful in deciding how the application would operate which aided the development process, particularly when issues appeared with the measurement technique and reference point finding methods.

5.1.2 Aim 2

The second aim was to implement the prototype application using identified and researched methods. This aim has been met using methods selected from those outlined in section 3 and the resulting application is documented in section 4.

5.1.3 Aim 3

Evaluate the success and appropriateness of the produced application

5.1.4 Aim 4

The final aim was to present conclusions about the project's successes and downfalls which is shown in the present section.

5.2 Comparison to other Products

5.3 Future Work

5.4 Self Appraisal

References

Adams, J., Khan, H. T., Raeside, R., & White, D. I. (2007). *Research methods for graduate business and social science students*. SAGE publications India.

Adobe. (2017). *Adobe photoshop.* Retrieved 02/04/2017, from http://www.adobe.com/uk/products/photoshop.html

Ahmad, I. S., Boufama, B., Habashi, P., Anderson, W., & Elamsy, T. (2015, Dec). Automatic license plate recognition: A comparative study. In *2015 ieee international symposium on signal processing and information technology (isspit)* (p. 635-640). doi: 10.1109/ISSPIT.2015.7394415

Aston, P. (2016). Sussmybike data acquisition - eurobike 2016. Retrieved 24/09/2016, from http://www.pinkbike.com/news/sussmybike-data-acquisition-eurobike-2016.html

Aydin, M. N., Harmsen, F., Van Slooten, K., & STEGWEE, R. (2004). An agile information systems development method in use. *Turkish Journal of Electrical Engineering & Computer Sciences*, *12*(2), 127–138.

Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., … others (2001). Manifesto for agile software development.

Benedict, T. (2012). Fox ird smartphone app sets ups shocks, forks for you. Retrieved 24/09/2016, from http://www.bikerumor.com/2012/08/29/fox-ird-smartphone-app-sets-ups-shocks-forks-for-you/

Dusty Dynamics. (2016). Introducing shockwiz. Retrieved 24/09/2016, from http://www.shockwiz.com/pages/shockwiz

European Science Foundation. (2007). *Medical imaging for improved patient care* [Policy Briefing]. Retrieved 03/10/2016, from http://www.esf.org/fileadmin/links/EMRC/ESF_POLICY28_V09_HD.pdf

Giant Manufacturing Co. Ltd. (2017a). *Giant fathom* [Product Listing]. Retrieved from https://www.giant-bicycles.com/gb/fathom-1

Giant Manufacturing Co. Ltd. (2017b). *Giant reign* [Product Listing]. Retrieved from https://www.giant-bicycles.com/gb/reign-1

Giant Manufacturing Co. Ltd. (2017c). *Giant stance* [Product Listing]. Retrieved from https://www.giant-bicycles.com/en-gb/bikes/model/stance/28553/99239/

Honda Motor Co. (2004). Honda develops world's first intelligent night vision system able to detect pedestrians and provide driver cautions-available on legend model to be released in fall 2004 [Press Release]. Retrieved 03/10/2016, from http://world.honda.com/news/2004/4040824a-eng.html

IMBA Europe. (2015). *Imba european mountain bike survey* (Survey). International Mountain Bike Association. Retrieved 24/09/2015, from http://www.imba-europe.com/sites/default/files/IMBA_INFOGRAPHIC_final.pdf

Bittner, K. (2002). *Use case modeling*. Addison-Wesley Longman Publishing Co., Inc.

Blake, D. F., Morris, R. V., Kocurek, G., Morrison, S. M., Downs, R. T., Bish, D., … Sarrazin, P. (2013). Curiosity at gale crater, mars: Characterization and analysis of the rocknest sand shadow. *Science*, *341*(6153). Retrieved from http://science.sciencemag.org/content/341/6153/1239505 doi: 10.1126/science.1239505

Bradski, G., & Kaehler, A. (2008). *Learning opencv: Computer vision with the opencv library*. " O'Reilly Media, Inc.".

Callaham, J. (2015). Google says there are now 1.4 billion active android devices worldwide. Retrieved 25/09/2016, from http://www.androidcentral.com/

google-says-there-are-now-14-billion-active-android-devices-worldwide

Cane Creek. (2016). *How to choose the right rear shock.* Retrieved 02/04/2017, from https://www.canecreek.com/culture/blog-news/how-to-choose-the-right-rear-shock

Cao, L., & Ramesh, B. (2008). Agile requirements engineering practices: An empirical study. *IEEE software*, *25*(1).

Cork, A., Justham, L., & West, A. (2012). Three-dimensional vision analysis to measure the release characteristics of elite bowlers in cricket. *Proceedings of the Institution of Mechanical Engineers, Part P: Journal of Sports Engineering and Technology*, 1754337112447264.

Diaz, K. M., Krupka, D. J., Chang, M. J., Peacock, J., Ma, Y., Goldsmith, J., … Davidson, K. W. (2015). Fitbit®: An accurate and reliable device for wireless physical activity tracking. *International Journal of Cardiology*, *185*, 138-140. Retrieved 24/09/2016, from http://dx.doi.org.ezproxy.napier.ac.uk/10.1016/j.ijcard.2015.03.038

Formosa, N. (2012). Social fitness apps storm cycling world. *Bicycle Retailer and Industry News*, *21*, 28-29. Retrieved 24/09/2016, from http://search.proquest.com.ezproxy.napier.ac.uk/docview/1018189163?rfr_id=info%3Axri%2Fsid%3Aprimo

Garvin, J. B., Malin, M. C., & Minitti, M. E. (2014, March 17). *Sedimentology of martain gravels from mardi twilight imaging: Techniques* (Tech. Rep. No. GSFC-E-DAA-TN13878). Retrieved 04/10/2016, from http://hdl.handle.net/2060/20150001290

Goodyear. (2014). Basic principles of air springs. Retrieved 02/04/2017, from http://www.lhtech.com/pdf/automation/pneumatic/Goodyear-Air-Spring-Data.pdf

Haralock, R. M., & Shapiro, L. G. (1991). *Computer and robot vision*. Addison-Wesley Longman Publishing Co., Inc.

Harker, J. (2010). German brand rose hopes to bloom in britain. Retrieved 24/09/2016, from http://www.bikebiz.com/news/read/german-brand-rose-hopes-to-bloom-in-britain/08815

Hwang, J. C. B. (2016, 05 19). *Shockwiz* (Trademark Application No. 87042899). 1000 West Fulton Market, 4th Floor, Chicago, Illinois, 60607, USA. Retrieved 25/09/2016, from https://trademarks.justia.com/870/42/shockwiz-87042899.html

Kim, S.-W., & Kim, N.-S. (2013). Dynamic characteristics of suspension bridge hanger cables using digital image processing. *NDT & E International*, *59*, 25 - 33. Retrieved from http://www.sciencedirect.com/science/article/pii/S0963869513000807 doi: http://dx.doi.org/10.1016/j.ndteint.2013.05.002

Kitchenham, B., Brereton, O. P., Budgen, D., Turner, M., Bailey, J., & Linkman, S. (2009). Systematic literature reviews in software engineering–a systematic literature review. *Information and software technology*, *51*(1), 7–15.

Klarica, A. (2001). Performance in motor sports. *British journal of sports medicine*, *35*(5), 290–291.

Kuhlman, D. (2009). *A python book: Beginning python, advanced python, and python exercises*. Dave Kuhlman.

Liszewski, A. (2016). Caterpillar's new s60 is the first smartphone with flir thermal imaging built right in. Retrieved 04/10/2016, from http://gizmodo.com/caterpillars-new-s60-is-the-first-smartphone-with-flir-1759685817

Martin, R. C. (2009). *Clean code: a handbook of agile software craftsmanship*. Pearson

Education.

Masad, E., Muhunthan, B., Shashidhar, N., & Harman, T. (1999). Internal structure characterization of asphalt concrete using image analysis. *Journal of Computing in Civil Engineering*, *13*(2), 88-95. Retrieved from http://dx.doi.org/10.1061/(ASCE)0887-3801(1999)13:2(88) doi: 10.1061/(ASCE)0887-3801(1999)13:2(88)

MatLab. (2017). *Edge detection.* Retrieved 02/04/2017, from https://uk.mathworks.com/discovery/edge-detection.html

Moeslund, T. B. (2012). *Introduction to video and image processing: Building real systems and applications*. Springer.

NASA JPL. (2012). *Contact instrument calibration targets on mars rover curiosity.* Retrieved from images-assets.nasa.gov/image/PIA15284/PIA15284~orig.jpg

OpenCV.org. (2017). *Opencv.* Retrieved 02/04/2017, from http://opencv.org/about.html

Potter, C. (2013). Ten years of land cover change on the california coast detected using landsat satellite image analysis: part 1—marin and san francisco counties. *Journal of Coastal Conservation*, *17*(4), 697–707. Retrieved from http://dx.doi.org/10.1007/s11852-013-0255-2 doi: 10.1007/s11852-013-0255-2

Pythonprogramming. (2016). *Pythonprogramming.net.* Retrieved 02/04/2017, from www.pythonprogramming.net

Rising, L., & Janoff, N. S. (2000). The scrum software development process for small teams. *IEEE software*, *17*(4), 26–32.

Rosebrock, A. (2017). *Pyimagesearch.* Retrieved 02/04/2017, from www.pyimagesearch.com

Rychlewski, J. (1984). On hooke's law. *Journal of Applied Mathematics and Mechanics*, *48*(3), 303–314.

Schröder, S., Meslin, P.-Y., Gasnault, O., Maurice, S., Cousin, A., Wiens, R., … Vaniman, D. (2015). Hydrogen detection with chemcam at gale crater. *Icarus*, *249*, 43 - 61. Retrieved from http://www.sciencedirect.com/science/article/pii/S001910351400445X (Special Issue: First Year of MSL) doi: http://dx.doi.org/10.1016/j.icarus.2014.08.029

Segers, J. (2008). *Analysis techniques for racecar data acquisition* (Tech. Rep.). SAE Technical Paper.

Shneiderman, B. (2010). *Designing the user interface: strategies for effective human-computer interaction*. Pearson Education India.

Staff, B. (2015). Yt industries launches consumer-direct sales in north america, oceania. Retrieved 24/09/2015, from http://www.bicycleretailer.com/industry-news/2015/01/30/yt-industries-launches-consumer-direct-sales-north-america-oceania

Titlestad, J., Fairlie-Clarke, T., Davie, M., Whittaker, A., & Grant, S. (2003). Experimental evaluation of mountain bike suspension systems. *Acta Polytechnica*, *43*, 15-20.

VersionOne. (2015). *9th annual state of agile survey.*

Wejer, J., Lendo, I., & Lewczuk, D. (2013). The effect of training on the jumping parameters of inexperienced warmblood horses in free jumping. *Journal of Equine Veterinary Science*, *33*(6), 483–486.

West, L. R. (2015). Strava: challenge yourself to greater heights in physical activity/cycling and running. *British Journal of Sports Medicine*, *49*, 1024. Retrieved 24/09/2016, from http://bjsm.bmj.com.ezproxy.napier.ac.uk/content/49/15/1024.full

Zuckerberg, M., Sittig, A., & Marlette, S. (2011, May 17). *Tagging digital media.* Google Patents. Retrieved from https://www.google.com/patents/US7945653 (US Patent 7,945,653)

A Android Experiments

A.1 Table of Android Experiments

Experiment	Purpose	Functional	Issues	Files	LOC
Hello CV	• Introduction to the android OpenCV library • Displays camera feed with fps	Yes	• Not fullscreen • Incorrect orientation	2	101
15Tile	• Sliding tile game • Uses camera feed as puzzle	Yes	• Not fullscreen	3	492
Blob Detection	• Demonstrates blob detection • Runs blob detection on tapped area from camera	No	• Crash on screen tap	3	311
Face Detection	• Detects faces in camera view • Puts boundary around detected faces	No	• Crash on load	3	279

A.2 Hello CV

```
package com.example.joe.base_app;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.util.Log;
import android.view.SurfaceView;
import android.widget.TextView;
import org.opencv.android.*;
import org.opencv.core.*;

public class MainActivity extends AppCompatActivity implements
↪ CameraBridgeViewBase.CvCameraViewListener2{

    private static String TAG = "Main Activity";
    private CameraBridgeViewBase mOpenCvCameraView;

    private BaseLoaderCallback mLoaderCallback = new BaseLoaderCallback(this) {
        @Override
        public void onManagerConnected(int status) {
            switch (status){
                case LoaderCallbackInterface.SUCCESS:{
                    Log.i(TAG, "Open CV loaded successfully");
                    mOpenCvCameraView.enableView();
                }break;
                default:{
                    super.onManagerConnected(status);
                }break;
            }
        }
    };

    @Override
    public void onResume(){
        super.onResume();
        OpenCVLoader.initAsync(OpenCVLoader.OPENCV_VERSION_3_1_0, this,
        ↪ mLoaderCallback);
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        Log.i(TAG, "called OnCreate");
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mOpenCvCameraView = (CameraBridgeViewBase) findViewById(R.id.HelloOpenCvView);
        mOpenCvCameraView.setVisibility(SurfaceView.VISIBLE);
        mOpenCvCameraView.setCvCameraViewListener(this);
    }

    @Override
    public void onPause(){
        super.onPause();
        if (mOpenCvCameraView != null){
            mOpenCvCameraView.disableView();
        }
    }
```

```java

    @Override
    public void onDestroy(){
        super.onDestroy();
        if (mOpenCvCameraView != null){
            mOpenCvCameraView.disableView();
        }
    }

    @Override
    public void onCameraViewStarted(int width, int height) {

    }

    @Override
    public void onCameraViewStopped() {

    }

    @Override
    public Mat onCameraFrame(CameraBridgeViewBase.CvCameraViewFrame inputFrame) {
        return inputFrame.rgba();
    }
}
```

A.3 15tile

```java
package com.example.joe.a15tile;

import android.annotation.SuppressLint;
import android.app.ActionBar;
import android.app.Activity;
import android.os.Bundle;
import android.os.Handler;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.view.MotionEvent;
import android.view.View;
import android.view.WindowManager;

import org.opencv.android.BaseLoaderCallback;
import org.opencv.android.CameraBridgeViewBase;
import org.opencv.android.JavaCameraView;
import org.opencv.android.LoaderCallbackInterface;
import org.opencv.android.OpenCVLoader;
import org.opencv.core.Mat;

public class MainActivity extends Activity implements
↪ CameraBridgeViewBase.CvCameraViewListener, View.OnTouchListener {
    private static final boolean AUTO_HIDE = true;
    private static final int AUTO_HIDE_DELAY_MILLIS = 3000;
    private static final int UI_ANIMATION_DELAY = 300;
    private final Handler mHideHandler = new Handler();
    private View mContentView;
    private final Runnable mHidePart2Runnable = new Runnable() {
```

```
        @SuppressLint("InlinedApi")
        @Override
        public void run() {
            // Delayed removal of status and navigation bar

            // Note that some of these constants are new as of API 16 (Jelly Bean)
            // and API 19 (KitKat). It is safe to use them, as they are inlined
            // at compile-time and do nothing on earlier devices.
//            mContentView.setSystemUiVisibility(View.SYSTEM_UI_FLAG_LOW_PROFILE
//                    | View.SYSTEM_UI_FLAG_FULLSCREEN
//                    | View.SYSTEM_UI_FLAG_LAYOUT_STABLE
//                    | View.SYSTEM_UI_FLAG_IMMERSIVE_STICKY
//                    | View.SYSTEM_UI_FLAG_LAYOUT_HIDE_NAVIGATION
//                    | View.SYSTEM_UI_FLAG_HIDE_NAVIGATION);
        }
    };
    private View mControlsView;
    private final Runnable mShowPart2Runnable = new Runnable() {
        @Override
        public void run() {
            // Delayed display of UI elements
            ActionBar actionBar = getActionBar();
            if (actionBar != null) {
                actionBar.show();
            }
            mControlsView.setVisibility(View.VISIBLE);
        }
    };
    private boolean mVisible;
    private final Runnable mHideRunnable = new Runnable() {
        @Override
        public void run() {
            hide();
        }
    };
    private final View.OnTouchListener mDelayHideTouchListener = new View.OnTouchListener() {
        @Override
        public boolean onTouch(View view, MotionEvent motionEvent) {
            if (AUTO_HIDE) {
                delayedHide(AUTO_HIDE_DELAY_MILLIS);
            }
            return false;
        }
    };


    private static final String TAG = "MainActivity";
    private CameraBridgeViewBase mOpenCvCameraView;
    private PuzzleProcessor mPuzzleProcessor;
    private MenuItem mItemHideNumbers;
    private MenuItem mItemStartNewGame;
    private int mGameWidth;
    private int mGameHeight;

    private BaseLoaderCallback mLoaderCallback = new BaseLoaderCallback(this) {
        @Override
        public void onManagerConnected(int status) {
```

```
            switch (status) {
                case LoaderCallbackInterface.SUCCESS: {
                    Log.i(TAG, "OpenCV loaded successfully");
                    mOpenCvCameraView.setOnTouchListener(MainActivity.this);
                    mOpenCvCameraView.enableView();
                }
                break;
                default: {
                    super.onManagerConnected(status);
                }
                break;
            }

        }
    };

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
        Log.d(TAG, "onCreate: Creating and setting view");
        mOpenCvCameraView = new JavaCameraView(this, -1);
        setContentView(mOpenCvCameraView);
        mOpenCvCameraView.setVisibility(CameraBridgeViewBase.VISIBLE);
        mOpenCvCameraView.setCvCameraViewListener(this);
        mPuzzleProcessor = new PuzzleProcessor();
        mPuzzleProcessor.prepareNewGame();
        mVisible = true;
        //mControlsView = findViewById(R.id.fullscreen_content_controls);
        //mContentView = findViewById(R.id.fullscreen_content);
        // Set up the user interaction to manually show or hide the system UI.
        //mContentView.setOnClickListener(new View.OnClickListener() {
        //    @Override
        //    public void onClick(View view) {
        //        toggle();
        //    }
        //});
        // Upon interacting with UI controls, delay any scheduled hide()
        // operations to prevent the jarring behavior of controls going away
        // while interacting with the UI.
        //findViewById(R.id.dummy_button).setOnTouchListener(mDelayHideTouchListener);
    }

    @Override
    public void onPause() {
        super.onPause();
        if (mOpenCvCameraView != null) mOpenCvCameraView.disableView();
    }

    @Override
    public void onResume() {
        super.onResume();
        if (!OpenCVLoader.initDebug()) {
            Log.d(TAG, "onResume: Internal OpenCV library not found. Using OpenCV
             ↪  manager for initialization");
            OpenCVLoader.initAsync(OpenCVLoader.OPENCV_VERSION_3_1_0, this,
             ↪  mLoaderCallback);
        } else {
```

```
            Log.d(TAG, "onResume: OpenCV lib found inside package. Using it!");
            mLoaderCallback.onManagerConnected(LoaderCallbackInterface.SUCCESS);
        }
    }

    @Override
    public void onDestroy() {
        super.onDestroy();
        if (mOpenCvCameraView != null) mOpenCvCameraView.disableView();
    }

    @Override
    protected void onPostCreate(Bundle savedInstanceState) {
        super.onPostCreate(savedInstanceState);

        // Trigger the initial hide() shortly after the activity has been
        // created, to briefly hint to the user that UI controls
        // are available.
        delayedHide(100);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        Log.i(TAG, "onCreateOptionsMenu: called");
        mItemHideNumbers = menu.add("Show/hide the tile numbers");
        mItemStartNewGame = menu.add("Start a new game");
        return true;
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        Log.i(TAG, "onOptionsItemSelected: Menu item selected" + item);
        if (item == mItemStartNewGame) {
            mPuzzleProcessor.prepareNewGame();
        } else if (item == mItemHideNumbers) {
            mPuzzleProcessor.toggleTileNumbers();
        }
        return true;
    }

    public void onCameraViewStarted(int width, int height) {
        mGameWidth = width;
        mGameHeight = height;
        mPuzzleProcessor.prepareGameSize(width, height);
    }

    public void onCameraViewStopped() {
    }

    public boolean onTouch(View view, MotionEvent event) {
        int xpos, ypos;

        xpos = (view.getWidth() - mGameWidth) / 2;
        xpos = (int) event.getX() - xpos;

        ypos = (view.getHeight() - mGameHeight) / 2;
        ypos = (int) event.getY() - ypos;

```

```java
        if (xpos >= 0 && xpos <= mGameWidth && ypos >= 0 && ypos <= mGameHeight)
            mPuzzleProcessor.deliverTouchEvent(xpos, ypos);

        return false;
    }

    public Mat onCameraFrame(Mat inputFrame) {
        return mPuzzleProcessor.puzzleFrame(inputFrame);
    }

    private void toggle() {
        if (mVisible) {
            hide();
        } else {
            show();
        }
    }

    private void hide() {
        // Hide UI first
        ActionBar actionBar = getActionBar();
        if (actionBar != null) {
            actionBar.hide();
        }
        //mControlsView.setVisibility(View.GONE);
        mVisible = false;

        // Schedule a runnable to remove the status and navigation bar after a delay
        mHideHandler.removeCallbacks(mShowPart2Runnable);
        mHideHandler.postDelayed(mHidePart2Runnable, UI_ANIMATION_DELAY);
    }

    @SuppressLint("InlinedApi")
    private void show() {
        // Show the system bar
        mContentView.setSystemUiVisibility(View.SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN
                | View.SYSTEM_UI_FLAG_LAYOUT_HIDE_NAVIGATION);
        mVisible = true;

        // Schedule a runnable to display UI elements after a delay
        mHideHandler.removeCallbacks(mHidePart2Runnable);
        mHideHandler.postDelayed(mShowPart2Runnable, UI_ANIMATION_DELAY);
    }

    /**
     * Schedules a call to hide() in [delay] milliseconds, canceling any
     * previously scheduled calls.
     */
    private void delayedHide(int delayMillis) {
        mHideHandler.removeCallbacks(mHideRunnable);
        mHideHandler.postDelayed(mHideRunnable, delayMillis);
    }
}
```

```java
package com.example.joe.a15tile;

/**
```

```
 * Created by joe on 07/11/16.
 */

import android.util.Log;

import org.opencv.core.*;
import org.opencv.imgproc.Imgproc;

public class PuzzleProcessor {

    private static final int GRID_SIZE = 4;
    private static final int GRID_AREA = GRID_SIZE * GRID_SIZE;
    private static final int GRID_EMPTY_INDEX = GRID_AREA - 1;
    private static final String TAG = "PuzzleProcessor";
    private static final Scalar GRID_EMPTY_COLOR = new Scalar(0x33, 0x33, 0x33, 0xFF);

    private int[] mIndexes;
    private int[] mTextWidths;
    private int[] mTextHeights;

    private Mat mRgba15;
    private Mat[] mCells15;
    private boolean mShowTileNumbers = true;

    public PuzzleProcessor() {
        mTextWidths = new int[GRID_AREA];
        mTextHeights = new int[GRID_AREA];
        mIndexes = new int[GRID_AREA];

        for (int i = 0; i < GRID_AREA; i++) mIndexes[i] = i;
    }

    public synchronized void prepareNewGame() {
        do {
            shuffle(mIndexes);
        } while (!isPuzzleSolvable());
    }

    public synchronized void prepareGameSize(int width, int height) {
        mRgba15 = new Mat(height, width, CvType.CV_8UC4);
        mCells15 = new Mat[GRID_AREA];

        for (int i = 0; i < GRID_SIZE; i++) {
            for (int j = 0; j < GRID_SIZE; j++) {
                int k = i * GRID_SIZE + j;
                mCells15[k] = mRgba15.submat(i * height / GRID_SIZE,
                        (i + 1) * height / GRID_SIZE,
                        j * width / GRID_SIZE,
                        (j + 1) * width / GRID_SIZE);
            }
        }

        for (int i = 0; i < GRID_AREA; i++) {
            Size s = Imgproc.getTextSize(Integer.toString(i + 1), 3, 1, 2, null);
            mTextHeights[i] = (int) s.height;
            mTextWidths[i] = (int) s.width;
        }
    }
```

```
    public synchronized Mat puzzleFrame(Mat inputPicture) {
        Mat[] cells = new Mat[GRID_AREA];
        int rows = inputPicture.rows();
        int cols = inputPicture.cols();

        rows = rows - rows % 4;
        cols = cols - cols % 4;

        for (int i = 0; i < GRID_SIZE; i++) {
            for (int j = 0; j < GRID_SIZE; j++) {
                int k = i * GRID_SIZE + j;
                cells[k] = inputPicture.submat(i * inputPicture.rows() / GRID_SIZE,
                        (i + 1) * inputPicture.rows() / GRID_SIZE,
                        j * inputPicture.cols() / GRID_SIZE,
                        (j + 1) * inputPicture.cols() / GRID_SIZE);
            }
        }

        rows = rows - rows % 4;
        cols = cols - cols % 4;

        for (int i = 0; i < GRID_AREA; i++) {
            int idx = mIndexes[i];
            if (idx == GRID_EMPTY_INDEX)
                mCells15[i].setTo(GRID_EMPTY_COLOR);
            else {
                cells[idx].copyTo(mCells15[i]);
                if (mShowTileNumbers) {
                    Imgproc.putText(mCells15[i],
                            Integer.toString(1 + idx),
                            new Point((cols / GRID_SIZE - mTextWidths[idx]) / 2,
                                    (rows / GRID_SIZE + mTextHeights[idx]) / 2),
                            3,
                            1,
                            new Scalar(255, 0, 0, 255),
                            2);
                }
            }
        }

        for (int i = 0; i < GRID_AREA; i++) cells[i].release();

        drawGrid(cols, rows, mRgba15);

        return mRgba15;
    }

    public void toggleTileNumbers() {
        mShowTileNumbers = !mShowTileNumbers;
    }

    public void deliverTouchEvent(int x, int y) {
        int rows = mRgba15.rows();
        int cols = mRgba15.cols();

        int row = (int) Math.floor(y * GRID_SIZE / rows);
        int col = (int) Math.floor(x * GRID_SIZE / cols);
```

```
        if (row < 0 || row >= GRID_SIZE || col < 0 || col >= GRID_SIZE) {
            Log.e(TAG, "deliverTouchEvent: Touch event ouside of image not expected");
            return;
        }

        int idx = row * GRID_SIZE + col;
        int idxToSwap = -1;

        if (idxToSwap < 0 && col > 0)
            if (mIndexes[idx - 1] == GRID_EMPTY_INDEX)
                idxToSwap = idx - 1;
        if (idxToSwap < 0 && col < GRID_SIZE - 1)
            if (mIndexes[idx + 1] == GRID_EMPTY_INDEX)
                idxToSwap = idx + 1;
        if (idxToSwap < 0 && row > 0)
            if (mIndexes[idx - GRID_SIZE] == GRID_EMPTY_INDEX)
                idxToSwap = idx - GRID_SIZE;
        if (idxToSwap < 0 && row < GRID_SIZE - 1)
            if (mIndexes[idx + GRID_SIZE] == GRID_EMPTY_INDEX)
                idxToSwap = idx + GRID_SIZE;

        if (idxToSwap >= 0) {
            synchronized (this) {
                int touched = mIndexes[idx];
                mIndexes[idx] = mIndexes[idxToSwap];
                mIndexes[idxToSwap] = touched;
            }
        }
    }

    private void drawGrid(int cols, int rows, Mat drawMat) {
        for (int i = 1; i < GRID_SIZE; i++) {
            Imgproc.line(drawMat,
                    new Point(0, i * rows / GRID_SIZE),
                    new Point(cols, i * rows / GRID_SIZE),
                    new Scalar(0, 255, 0, 255),
                    3);
            Imgproc.line(drawMat,
                    new Point(i * cols / GRID_SIZE, rows),
                    new Point(i * cols / GRID_SIZE, rows),
                    new Scalar(0, 255, 0, 255),
                    3);
        }
    }

    private static void shuffle(int[] array) {
        for (int i = array.length; i > 1; i--) {
            int temp = array[i - 1];
            int randIx = (int) (Math.random() * i);
            array[i - 1] = array[randIx];
            array[randIx] = temp;
        }
    }

    private boolean isPuzzleSolvable() {
        int sum = 0;
        for (int i = 0; i < GRID_AREA; i++) {
```

```
            if (mIndexes[i] == GRID_EMPTY_INDEX) sum += (i / GRID_SIZE) + 1;
            else {
                int smaller = 0;
                for (int j = i+1; j < GRID_AREA; j++){
                    if (mIndexes[j] < mIndexes[i]) smaller++;
                }
                sum += smaller;
            }
        }
        return sum % 2 == 0;
    }

}
```

A.4 BLOB Analysis

```
package com.example.joe.blob_analysis;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;
import android.view.MotionEvent;
import android.view.SurfaceView;
import android.view.View;
import android.view.Window;
import android.view.WindowManager;

import org.opencv.android.BaseLoaderCallback;
import org.opencv.android.CameraBridgeViewBase;
import org.opencv.android.LoaderCallbackInterface;
import org.opencv.android.OpenCVLoader;
import org.opencv.core.Core;
import org.opencv.core.CvType;
import org.opencv.core.Mat;
import org.opencv.core.MatOfPoint;
import org.opencv.core.Rect;
import org.opencv.core.Scalar;
import org.opencv.core.Size;
import org.opencv.imgproc.Imgproc;

import java.util.List;

/**
 * Created by joe on 12/11/16.
 */

public class ColorBlobDetectionActivity extends Activity implements View.OnTouchListener, CameraBridgeViewBase.CvCameraViewListener2 {
    private static final String TAG = "Activity: ";

    private boolean mIsColorSelected = false;
    private Mat mRgba;
    private Scalar mBlobColorRgba;
    private Scalar mBlobColorHsv;
    private ColorBlobDetector mDetector;
    private Mat mSpectrum;
```

```java
    private Size SPECTRUM_SIZE;
    private Scalar CONTOUR_COLOR;

    private CameraBridgeViewBase mOpenCvCameraView;

    private BaseLoaderCallback mLoaderCallback = new BaseLoaderCallback(this) {
        @Override
        public void onManagerConnected(int status) {
            switch (status) {
                case LoaderCallbackInterface.SUCCESS: {
                    Log.i(TAG, "onManagerConnected: OpenCV Loaded");
                    mOpenCvCameraView.enableView();
                mOpenCvCameraView.setOnTouchListener(ColorBlobDetectionActivity.this);
                }
                break;
                default: {
                    super.onManagerConnected(status);
                }
                break;
            }
        }
    };

    public ColorBlobDetectionActivity() {
        Log.i(TAG, "ColorBlobDetectionActivity: Instantiated new");
    }

    @Override
    public void onCreate(Bundle savedInstanceState) {
        Log.i(TAG, "onCreate: Called");
        super.onCreate(savedInstanceState);
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
        setContentView(R.layout.color_blob_detection_surface_view);
        mOpenCvCameraView = (CameraBridgeViewBase)
         ↪  findViewById(R.id.color_blob_detection_activity_surface_view);
        mOpenCvCameraView.setVisibility(SurfaceView.VISIBLE);
        mOpenCvCameraView.setCvCameraViewListener(this);
    }

    @Override
    public void onPause() {
        super.onPause();
        if (mOpenCvCameraView != null) {
            mOpenCvCameraView.disableView();
        }
    }

    @Override
    public void onResume() {
        super.onResume();
        if (!OpenCVLoader.initDebug()) {
            Log.d(TAG, "onResume: OpenCV not found. Using Manager");
            OpenCVLoader.initAsync(OpenCVLoader.OPENCV_VERSION_3_1_0, this,
             ↪  mLoaderCallback);
        } else {
            Log.d(TAG, "onResume: OpenCV found in package.");
            mLoaderCallback.onManagerConnected(LoaderCallbackInterface.SUCCESS);
```

```
        }
    }

    public void onDestroy() {
        super.onDestroy();
        if (mOpenCvCameraView != null) {
            mOpenCvCameraView.disableView();
        }
    }

    public void onCameraViewStarted(int width, int height) {
        mRgba = new Mat(height, width, CvType.CV_8UC4);
        mDetector = new ColorBlobDetector();
        mSpectrum = new Mat();
        mBlobColorRgba = new Scalar(255);
        mBlobColorHsv = new Scalar(255);
        SPECTRUM_SIZE = new Size(200, 64);
        CONTOUR_COLOR = new Scalar(255, 0, 0, 255);
    }

    public void onCameraViewStopped() {
        mRgba.release();
    }

    public boolean onTouch(View v, MotionEvent event){
        int cols = mRgba.cols();
        int rows = mRgba.rows();

        int xOffset = (mOpenCvCameraView.getWidth() - cols) / 2;
        int yOffset = (mOpenCvCameraView.getHeight() - rows) / 2;

        int x = (int)event.getX() - xOffset;
        int y = (int) event.getY() - yOffset;

        Log.i(TAG, "onTouch: Touch coordinates: (" + x + ", " + y + ")");

        if ((x < 0) || (y < 0) || (x > cols) || (y > rows)) return false;

        Rect touchedRect = new Rect();

        touchedRect.x = (x > 4) ? x - 4 : 0;
        touchedRect.y = (y > 4) ? y - 4 : 0;

        touchedRect.width = (x + 4 < cols) ? x + 4 - touchedRect.x : cols -
         ↪  touchedRect.x;
        touchedRect.height = (y + 4 < rows) ? y + 4 - touchedRect.y : rows -
         ↪  touchedRect.y;

        Mat touchedRegionRgba = mRgba.submat(touchedRect);

        Mat touchedRegionHsv = new Mat();
        Imgproc.cvtColor(touchedRegionRgba, touchedRegionHsv,
         ↪  Imgproc.COLOR_RGB2HSV_FULL);

        mBlobColorHsv = Core.sumElems(touchedRegionHsv);
        int pointCount = touchedRect.width * touchedRect.height;
        for (int i = 0; i < mBlobColorHsv.val.length; i++) {
            mBlobColorHsv.val[i] /= pointCount;
```

```
        }

        mBlobColorRgba = convertScalarHsv2Rgba(mBlobColorHsv);

        Log.i(TAG, "onTouch: Touched rgba color: (" + mBlobColorRgba.val[0] + ", " +
         ↪  mBlobColorRgba.val[1] +", " + mBlobColorRgba.val[2] + ", " +
         ↪  mBlobColorRgba.val[3] + ")");

        mDetector.setHsvColor(mBlobColorHsv);

        Imgproc.resize(mDetector.getSpectrum(), mSpectrum, SPECTRUM_SIZE);
        mIsColorSelected = true;

        touchedRegionRgba.release();
        touchedRegionHsv.release();

        return false;
    }

    public Mat onCameraFrame(CameraBridgeViewBase.CvCameraViewFrame inputFrame) {
        mRgba = inputFrame.rgba();
        if (mIsColorSelected) {
            mDetector.process(mRgba);
            List<MatOfPoint> contours = mDetector.getContours();
            Log.e(TAG, "onCameraFrame: Contours count: " + contours.size());
            Imgproc.drawContours(mRgba, contours, -1, CONTOUR_COLOR);
            Mat colorLabel = mRgba.submat(4, 68, 4, 68);
            colorLabel.setTo(mBlobColorRgba);
            Mat spectrumLabel = mRgba.submat(4, 4 + mSpectrum.rows(), 70, 70 +
             ↪  mSpectrum.cols());
            mSpectrum.copyTo(spectrumLabel);
        }

        return mRgba;
    }

    private Scalar convertScalarHsv2Rgba(Scalar hsvColor) {
        Mat pointMatRgba = new Mat();
        Mat pointMatHsv = new Mat(1, 1, CvType.CV_8UC3, hsvColor);
        Imgproc.cvtColor(pointMatHsv, pointMatRgba, Imgproc.COLOR_HSV2RGB_FULL, 4);
        return new Scalar(pointMatRgba.get(0, 0));
    }

}
```

```
package com.example.joe.blob_analysis;

import org.opencv.core.Core;
import org.opencv.core.CvType;
import org.opencv.core.Mat;
import org.opencv.core.MatOfPoint;
import org.opencv.core.Scalar;
import org.opencv.imgproc.Imgproc;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
```

```

/**
 * Created by joe on 12/11/16.
 */

public class ColorBlobDetector {

    private Scalar mLowerBound = new Scalar(0);
    private Scalar mUpperBound = new Scalar(0);

    private static double mMinContourArea = 0.1;

    private Scalar mColorRadius = new Scalar(25, 50, 50, 0);
    private Mat mSpectrum = new Mat();
    private List<MatOfPoint> mContours = new ArrayList<MatOfPoint>();

    Mat mPyrDownMat = new Mat();
    Mat mHsvMat = new Mat();
    Mat mMask = new Mat();
    Mat mDilatedMask = new Mat();
    Mat mHierarchy = new Mat();

    public void setColorRadius(Scalar radius) {
        mColorRadius = radius;
    }

    public void setHsvColor(Scalar hsvColor) {
        double minH = (hsvColor.val[0] >= mColorRadius.val[0]) ? hsvColor.val[0] - mColorRadius.val[0] : 0;
        double maxH = (hsvColor.val[0] + mColorRadius.val[0] <= 255) ? hsvColor.val[0] + mColorRadius.val[0] : 255;

        mLowerBound.val[0] = minH;
        mUpperBound.val[0] = maxH;

        mLowerBound.val[1] = hsvColor.val[1] - mColorRadius.val[1];
        mUpperBound.val[1] = hsvColor.val[1] + mColorRadius.val[1];

        mLowerBound.val[2] = hsvColor.val[2] - mColorRadius.val[2];
        mUpperBound.val[2] = hsvColor.val[2] + mColorRadius.val[2];

        mLowerBound.val[3] = 0;
        mUpperBound.val[3] = 255;

        Mat spectrumHsv = new Mat(1, (int)(maxH-minH), CvType.CV_8UC3);

        for (int j = 0; j < maxH-minH; j++) {
            byte[] tmp = {(byte)(minH+j), (byte)255, (byte)255};
            spectrumHsv.put(0, j, tmp);
        }

        Imgproc.cvtColor(spectrumHsv, mSpectrum, Imgproc.COLOR_HSV2RGB_FULL);
    }

    public Mat getSpectrum() {
        return mSpectrum;
    }

```

```java
    public void setMinContourArea(double area){mMinContourArea = area;}

    public void process(Mat rgbaImage){
        Imgproc.pyrDown(rgbaImage, mPyrDownMat);
        Imgproc.pyrDown(mPyrDownMat, mPyrDownMat);

        Imgproc.cvtColor(mPyrDownMat, mHsvMat, Imgproc.COLOR_RGB2HSV_FULL);

        Core.inRange(mHsvMat, mLowerBound, mUpperBound, mMask);
        Imgproc.dilate(mMask, mDilatedMask, new Mat());

        List<MatOfPoint> contours = new ArrayList<MatOfPoint>();

        Imgproc.findContours(mDilatedMask, contours, mHierarchy,
         ↪  Imgproc.RETR_EXTERNAL, Imgproc.CHAIN_APPROX_SIMPLE);

        double maxArea = 0;
        Iterator<MatOfPoint> each = contours.iterator();
        while (each.hasNext()) {
            MatOfPoint wrapper = each.next();
            double area = Imgproc.contourArea(wrapper);
            if (area > maxArea) {
                maxArea = area;
            }
        }

        mContours.clear();
        each = contours.iterator();
        while (each.hasNext()) {
            MatOfPoint contour = each.next();
            if (Imgproc.contourArea(contour) > mMinContourArea*maxArea) {
                Core.multiply(contour, new Scalar(4,4), contour);
                mContours.add(contour);
            }
        }
    }

    public List<MatOfPoint> getContours(){return mContours;}

}
```

A.5 Face Detection

```java
package com.example.joe.face_recognition;

import org.opencv.core.Mat;
import org.opencv.core.MatOfRect;

/**
 * Created by joe on 08/11/16.
 */

public class DetectionBasedTracker {
    public DetectionBasedTracker(String cascadeName, int minFaceSize) {
        mNativeObj = nativeCreateObject(cascadeName, minFaceSize);
    }
```

```java

    public void start() {
        nativeStart(mNativeObj);
    }

    public void stop() {
        nativeStop(mNativeObj);
    }

    public void setMinFaceSize(int size) {
        nativeSetFaceSize(mNativeObj, size);
    }

    public void detect(Mat imageGray, MatOfRect faces) {
        nativeDetect(mNativeObj, imageGray.getNativeObjAddr(),
         ↪  faces.getNativeObjAddr());
    }

    public void release() {
        nativeDestroyObject(mNativeObj);
        mNativeObj = 0;
    }

    private long mNativeObj = 0;

    private static native long nativeCreateObject(String cascadeName, int minFaceSize);
    private static native void nativeDestroyObject(long thiz);
    private static native void nativeStart(long thiz);
    private static native void nativeStop(long thiz);
    private static native void nativeSetFaceSize(long thiz, int size);
    private static native void nativeDetect(long thiz, long inputImage, long faces);
}
```

```java
package com.example.joe.face_recognition;

import android.app.Activity;
import android.content.Context;
import android.os.Bundle;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.view.WindowManager;

import org.opencv.android.BaseLoaderCallback;
import org.opencv.android.CameraBridgeViewBase;
import org.opencv.android.LoaderCallbackInterface;
import org.opencv.android.OpenCVLoader;
import org.opencv.core.Mat;
import org.opencv.core.MatOfRect;
import org.opencv.core.Rect;
import org.opencv.core.Scalar;
import org.opencv.core.Size;
import org.opencv.imgproc.Imgproc;
import org.opencv.objdetect.CascadeClassifier;

import java.io.File;
import java.io.FileOutputStream;
```

```java
import java.io.IOException;
import java.io.InputStream;

/**
 * Created by joe on 08/11/16.
 */

public class FrActivity extends Activity implements
 ↪  CameraBridgeViewBase.CvCameraViewListener2 {

    private static final String TAG             = "FrActivity";
    private static final Scalar FACE_RECT_COLOR = new Scalar(0,255,0,255);
    private static final int    JAVA_DETECTOR   = 0;
    private static final int    NATIVE_DETECTOR = 1;

    private MenuItem mItemFace50;
    private MenuItem mItemFace40;
    private MenuItem mItemFace30;
    private MenuItem mItemFace20;
    private MenuItem mItemType;

    private Mat                    mRgba;
    private Mat                    mGray;
    private File                   mCascadeFile;
    private CascadeClassifier      mJavaDetector;
    private DetectionBasedTracker  mNativeDetector;

    private int         mDetectorType = JAVA_DETECTOR;
    private String[]    mDetectorName;

    private float   mRelativeFaceSize = 0.2f;
    private int     mAbsoluteFaceSize = 0;

    private CameraBridgeViewBase mOpenCvCameraView;

    private BaseLoaderCallback mLoaderCallback = new BaseLoaderCallback(this) {
        @Override
        public void onManagerConnected(int status) {
            switch (status){
                case LoaderCallbackInterface.SUCCESS:{
                    Log.i(TAG, "onManagerConnected: OpenCV loaded correctly");
                    System.loadLibrary("detection_based_tracker");
                    try{
                        InputStream is =
                         ↪  getResources().openRawResource(R.raw.lbpcascade_frontalface);
                        File cascadeDir = getDir("cascade", Context.MODE_PRIVATE);
                        mCascadeFile = new File(cascadeDir,
                         ↪  "lbpcascade_frontalface.xml");
                        FileOutputStream os = new FileOutputStream(mCascadeFile);

                        byte[] buffer = new byte[4096];
                        int bytesRead;
                        while ((bytesRead = is.read(buffer))!=-1){
                            os.write(buffer, 0, bytesRead);
                        }
                        is.close();
                        os.close();

```

```
                        mJavaDetector = new CascadeClassifier(mCascadeFile.getAbsolutePath());
                        if (mJavaDetector.empty()){
                            Log.e(TAG, "onManagerConnected: Failed to load cascade classifier");
                            mJavaDetector = null;
                        }else{
                            Log.i(TAG, "onManagerConnected: Loaded cascade classifier from "+mCascadeFile.getAbsolutePath());
                        }
                        mNativeDetector = new DetectionBasedTracker(mCascadeFile.getAbsolutePath(), 0);
                        cascadeDir.delete();
                    }catch (IOException e){
                        e.printStackTrace();
                        Log.e(TAG, "onManagerConnected: Failed to load cascade. Exception thrown: "+e);
                    }

                    mOpenCvCameraView.enableView();
                }break;
                default:{
                    super.onManagerConnected(status);
                }break;
            }
        }
    };

    public FrActivity(){
        mDetectorName = new String[2];
        mDetectorName[JAVA_DETECTOR]   = "Java";
        mDetectorName[NATIVE_DETECTOR] = "Native (tracking)";
        Log.i(TAG, "FrActivity: Instantiated new" + this.getClass());
    }

    @Override
    public void onCreate(Bundle savedInstanceState){
        Log.i(TAG, "onCreate: Called");
        super.onCreate(savedInstanceState);
        getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
        setContentView(R.layout.face_detect_surface_view);
        mOpenCvCameraView = (CameraBridgeViewBase) findViewById(R.id.fd_activity_surface_view);
        mOpenCvCameraView.setVisibility(CameraBridgeViewBase.VISIBLE);
        mOpenCvCameraView.setCvCameraViewListener(this);
    }

    @Override
    public void onPause(){
        super.onPause();
        if (mOpenCvCameraView != null) mOpenCvCameraView.disableView();
    }

    @Override
    public void onResume(){
        super.onResume();
        if (!OpenCVLoader.initDebug()){
```

```
            Log.d(TAG, "onResume: Internal OpenCV lib not found. Using OpenCV manager
             ↪ for initialization");
             ↪ OpenCVLoader.initAsync(OpenCVLoader.OPENCV_VERSION_3_1_0,this,mLoaderCallback);
        }else{
            Log.d(TAG, "onResume: OpenCV lib found inside package.");
            mLoaderCallback.onManagerConnected(LoaderCallbackInterface.SUCCESS);
        }
    }

    public void onDestroy(){
        super.onDestroy();
        mOpenCvCameraView.disableView();
    }

    public void onCameraViewStarted(int width, int height){
        mGray = new Mat();
        mRgba = new Mat();
    }

    public void onCameraViewStopped(){
        mGray.release();
        mRgba.release();
    }

    public Mat onCameraFrame(CameraBridgeViewBase.CvCameraViewFrame inputFrame){
        mRgba = inputFrame.rgba();
        mGray = inputFrame.gray();

        if (mAbsoluteFaceSize == 0){
            int height = mGray.rows();
            if (Math.round(height * mRelativeFaceSize)>0){
                mAbsoluteFaceSize = Math.round(height * mRelativeFaceSize);
            }
            mNativeDetector.setMinFaceSize(mAbsoluteFaceSize);
        }

        MatOfRect faces = new MatOfRect();

        if (mDetectorType == JAVA_DETECTOR){
            if (mJavaDetector != null){
                mJavaDetector.detectMultiScale(mGray,
                        faces,
                        1.1,
                        2,
                        2,
                        new Size(mAbsoluteFaceSize, mAbsoluteFaceSize),
                        new Size());
            }
        }else if (mDetectorType == NATIVE_DETECTOR){
            if (mNativeDetector != null){
                mNativeDetector.detect(mGray, faces);
            }
        }else{
            Log.e(TAG, "onCameraFrame: Detection method is not selected");
        }

        Rect[] facesArray = faces.toArray();
```

```
        for (Rect aFacesArray : facesArray)
            Imgproc.rectangle(mRgba, aFacesArray.tl(), aFacesArray.br(),
             ↪  FACE_RECT_COLOR, 3);

        return mRgba;
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu){
        Log.i(TAG, "onCreateOptionsMenu: called");
        mItemFace50 = menu.add("Face size 50%");
        mItemFace40 = menu.add("Face size 40%");
        mItemFace30 = menu.add("Face size 30%");
        mItemFace20 = menu.add("Face size 20%");
        mItemType   = menu.add(mDetectorName[mDetectorType]);
        return true;
    }

    private void setMinFaceSize(float faceSize){
        mRelativeFaceSize = faceSize;
        mAbsoluteFaceSize = 0;
    }

    private void setDetectorType(int type){
        if (mDetectorType != type){
            mDetectorType = type;
            if (type == NATIVE_DETECTOR){
                Log.i(TAG, "setDetectorType: Detection based tracker enabled");
                mNativeDetector.start();
            }else{
                Log.i(TAG, "setDetectorType: Cascade detector enabled");
                mNativeDetector.stop();
            }
        }
    }
}
```

B Python Experiments

B.1 Table of Python Experiments

Experiment	Purpose	Functional	Issues	Files	LOC
Find Game	• Identify red game cartridge out of 3 • Displays boundary around correct part of image	Yes	N/A	1	18
Threshold Methods	• Demonstrates various thresholding methods on an image • Original image text unreadable but clear after techniques are applied	Yes	N/A	1	14
Image Operations	• Moves parts of an image to other locations using arrays • Demonstrates how pixel data is stored	Yes	N/A	1	15
Distance to Camera	• Calculates distance to the camera from an identified object • Displays various distances using 3 images	Yes	N/A	1	38

B.2 find_game.py

```
import numpy as np
import cv2

image = cv2.imread('games.jpg')

upper = np.array([65, 65, 255])
lower = np.array([0, 0, 200])
mask = cv2.inRange(image, lower, upper)

cnts, _ = cv2.findContours(mask.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
c = max(cnts, key=cv2.contourArea)

peri = cv2.arcLength(c, True)
approx = cv2.approxPolyDP(c, 0.05 * peri, True)

cv2.drawContours(image, [approx], -1, (0,255,0), 4)
cv2.imshow('Image', image)
cv2.waitKey(0)
```

B.3 thresholding.py

```
import cv2
import numpy as np

img = cv2.imread('bookpage.jpg')
grayscaled = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
_, threshold = cv2.threshold(img, 12, 255, cv2.THRESH_BINARY)
_, gs_threshold = cv2.threshold(grayscaled, 10, 255, cv2.THRESH_BINARY)
adaptive = cv2.adaptiveThreshold(grayscaled, 255, cv2.ADAPTIVE_THRESH_GAUSSIAN_C,
    cv2.THRESH_BINARY, 115, 1)
cv2.imshow('original', img)
cv2.imshow('threshold', threshold)
cv2.imshow('grayscaled', gs_threshold)
cv2.imshow('adaptive', adaptive)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

B.4 img_ops.py

```
import cv2
import numpy as np

img = cv2.imread('watch.jpg', cv2.IMREAD_COLOR)
px = img[55, 55]
img[55, 55] = [255, 255, 255]
px = img[55, 55]
print(px)
px = img[100:150, 100:150]
print(px)

watch_face = img[37:111, 107:194]
```

```python
img[0:74, 0:87] = watch_face
cv2.imshow('image', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

B.5 distance_to_camera.py

```python
import numpy as np
import cv2


def find_marker(image):
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    gray = cv2.GaussianBlur(gray, (5, 5), 0)
    edged = cv2.Canny(gray, 35, 125)

    (cnts, _) = cv2.findContours(edged.copy(), cv2.RETR_LIST, cv2.CHAIN_APPROX_SIMPLE)
    c = max(cnts, key=cv2.contourArea)

    return cv2.minAreaRect(c)


def distance_to_camera(knownWidth, focalLength, perWidth):
    return (knownWidth * focalLength) / perWidth


KNOWN_DISTANCE = 24.0
KNOWN_WIDTH = 11.0
IMAGE_PATHS = ['2ft.png', '3ft.png', '4ft.png']

image = cv2.imread(IMAGE_PATHS[0])
marker = find_marker(image)
focalLength = (marker[1][0] * KNOWN_DISTANCE) / KNOWN_WIDTH

for imagePath in IMAGE_PATHS:
    image = cv2.imread(imagePath)
    marker = find_marker(image)
    inches = distance_to_camera(KNOWN_WIDTH, focalLength, marker[1][0])

    box = np.int0(cv2.cv.BoxPoints(marker))
    cv2.drawContours(image, [box], -1, (0, 255, 0), 2)
    cv2.putText(image, '%.2fft' % (inches/12),
                (image.shape[1] - 200, image.shape[0] -20), cv2.FONT_HERSHEY_SIMPLEX,
                ↪ 2.0, (0, 255, 0), 3)
    cv2.imshow('image', image)
    cv2.waitKey(0)
```

C Development Log

D EXIF Extraction Code

```python
def get_sensor_size(exif):
    # (Resolution in pixels / Focal plane resolution in dpi) X 25.4(mm / in) = size in
    ↪ mm
    # Do for hor and ver
    horizontal_measurement = None
    vertical_measurement = None


def get_focal_length(exif):
    """

    :param exif:
    :return:
    """
    measurement, divisor = exif['FocalLength']
    return float(measurement)/float(divisor)


def get_exif(image_path):
    """

    :param image_path:
    :return:
    """
    image = PIL.Image.open(image_path)
    exif = {PIL.ExifTags.TAGS[k]: v
            for k, v in image._getexif().items()
            if k in PIL.ExifTags.TAGS}
    return exif
```

E Commit Log

F Photos

G Source Code

G.1 main.py

```python
#! /usr/bin/env python

import argparse
import sys
import warnings
from image_processor import ImageProcessor
from pressure_calculator import PressureCalculator

```

```
warnings.filterwarnings('ignore')


def main():
    parser = argparse.ArgumentParser()
    parser.add_argument('-i',
                        '--image',
                        type=str,
                        nargs=2,
                        action='store',
                        metavar='',
                        help='The path to the image you wish to use')
    parser.add_argument('-c',
                        '--colour',
                        type=str,
                        action='store',
                        metavar='',
                        help='The colour of the o-ring on the shock')
    parser.add_argument('-s',
                        '--sag',
                        type=int,
                        action='store',
                        metavar='',
                        help='The desired sag percentage')
    parser.add_argument('-p',
                        '--pressure',
                        type=int,
                        action='store',
                        metavar='',
                        help='The pressure you have already put into your bike')
    parser.add_argument('-t',
                        '--stroke',
                        type=float,
                        action='store',
                        metavar='',
                        help='')
    parser.add_argument('-d',
                        '--debug',
                        action='store_true')

    args = parser.parse_args()
    if args.image is None:
        print 'This program requires an image'
        parser.print_help()
        sys.exit(1)
    if args.colour is None:
        print 'Please specify a colour for the o-ring'
        parser.print_help()
        sys.exit(1)

    image_processor = ImageProcessor(args.colour, args.debug)
    pressure_calculator = PressureCalculator(args.sag, args.stroke, args.debug)
    pressure_calculator.measurement_100 =
    ↪  image_processor.get_measurement(args.image[0])
    image_processor = ImageProcessor(args.colour, args.debug)
    pressure_calculator.measurement_150 =
    ↪  image_processor.get_measurement(args.image[1])
    #pressure_calculator.measurement_100 = 30.0
```

```
    #pressure_calculator.measurement_150 = 20.0
    setting = pressure_calculator.calculate()
    print '\nYour recommended pressure setting is'
    print str(round(setting, 0)) + ' psi'

if __name__ == '__main__':
    main()
```

G.2 image_processor.py

```
import cv2
import numpy
import sys
import utils
from sklearn.cluster import MiniBatchKMeans


def show_image(image, wait_time):
    try:
        cv2.imshow(str(image), image)
        cv2.waitKey(wait_time)
        return 0
    except cv2.error as e:
        print e.message


class ImageProcessor:
    REF_POINT_KNOWN_WIDTH = 9.5

    _image_path = None
    _edged_image = None
    _debug = False
    _ref_point = None
    _oring = None
    _colour = None

    def __init__(self, colour='red', debug=False):
        self._colour = colour
        self._debug = debug

    def _edge_detect(self):
        image = cv2.imread(self._image_path)
        gray_scaled = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        blurred = cv2.GaussianBlur(gray_scaled, (5, 5), 0)
        edged = cv2.Canny(blurred, 10, 90, apertureSize=3)
        return edged

    def _quantify_colors(self):
        image = cv2.imread(self._image_path)
        (h, w) = image.shape[:2]
        image = cv2.cvtColor(image, cv2.COLOR_BGR2LAB)
        image = image.reshape((image.shape[0] * image.shape[1], 3))
        clt = MiniBatchKMeans(n_clusters=8)
        labels = clt.fit_predict(image)
        quantified = clt.cluster_centers_.astype('uint8')[labels]
        quantified = quantified.reshape((h, w, 3))
```

```
        return cv2.cvtColor(quantified, cv2.COLOR_LAB2BGR)

    def _find_red(self):
        image = self._quantify_colors()
        upper_bound = numpy.array([100, 100, 255])
        lower_bound = numpy.array([0, 0, 130])
        mask = cv2.inRange(image, lower_bound, upper_bound)
        marker = cv2.bitwise_and(image, image, mask=mask)
        marker = cv2.cvtColor(marker, cv2.COLOR_BGR2GRAY)
        contours, _ = cv2.findContours(marker.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
        return sorted(contours, key=cv2.contourArea, reverse=True)[:2]

    def _find_ref(self):
        try:
            self._ref_point = self._find_red()[0]
        except (ValueError, IndexError):
            print 'Could not find reference point\n' \
                  'Is the the correct colour given and is one present in the image?\nIf ' \
                  'yes, please try again.'
            sys.exit(1)

    def _find_oring(self):
        if self._colour == 'red':
            try:
                self._oring = self._find_red()[1]
            except IndexError:
                print 'Could not find o-ring\n'\
                      'Is the the correct colour given and is one present in the image?\nIf ' \
                      'yes, please try again.'
        elif self._colour == 'black':
            image = cv2.imread(self._image_path)
            (height, width) = image.shape[:2]
            gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
            (T, thresh) = cv2.threshold(gray, 30, 150, cv2.THRESH_BINARY)
            contours, _ = cv2.findContours(thresh.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
            contours = sorted(contours, key=cv2.contourArea, reverse=True)[:5]
            for contour in contours:
                _, y, _, _ = cv2.boundingRect(contour)
                if y > (height / 3):
                    self._oring = contour

    def _get_ref_point_width(self):
        (_, _), radius = cv2.minEnclosingCircle(self._ref_point)
        if self._debug:
           utils.debug_print(self.__class__.__name__, '_get_ref_point_width', radius * 2)
        return radius * 2

    def _get_oring_height(self):
        _, y, _, h = cv2.boundingRect(self._oring)
        if self._debug:
            utils.debug_print(self.__class__.__name__, '_get_oring_height', y+h)
        return y + h
```

```python
    def _get_measurement_px(self):
        min_line_length = 1000
        max_line_gap = 10
        image_height, image_width = self._edged_image.shape[:2]
        y_limit = self._get_oring_height()
        y_min = image_height
        y_max = 0
        lines = cv2.HoughLinesP(self._edged_image,
                                1,
                                numpy.pi / 180,
                                50,
                                min_line_length,
                                max_line_gap)
        for x1, y1, x2, y2 in lines[0]:
            if x1 >= image_width / 2 and \
                            y1 >= image_height / 3 and \
                            y2 <= y_limit and \
                            abs(x1 - x2) <= 1:
                y_min = min(y_min, y1)
                y_max = max(y_max, y2)

        if self._debug:
            utils.debug_print(self.__class__.__name__, '_get_measurement_px', y_max - y_min)
        return y_max - y_min

    def _convert_px_mm(self, measurement_px):
        pixels_per_mm = self._get_ref_point_width() / self.REF_POINT_KNOWN_WIDTH
        if self._debug:
           utils.debug_print(self.__class__.__name__, '_convert_px_mm', pixels_per_mm)
        return measurement_px / pixels_per_mm

    def get_measurement(self, image_path):
        self._image_path = image_path
        if self._debug:
            utils.debug_print(self.__class__.__name__, 'filepath', self._image_path)
        self._edged_image = self._edge_detect()
        self._find_ref()
        self._find_oring()
        measurement_px = self._get_measurement_px()
        measurement_mm = self._convert_px_mm(measurement_px)

        if self._debug:
            utils.debug_print(self.__class__.__name__, 'get_measurement', measurement_mm)
        if measurement_mm <= 0:
            print 'Incorrect measurement produced from image {i}\n'\
                  'Please check settings and try again.'.format(i=self._image_path)
            sys.exit(1)
        return measurement_mm
```

G.3 pressure_calculator.py

```python
import numpy as np
import scipy.stats as stats
import utils
```

```python


class PressureCalculator:

    measurement_100 = None
    measurement_150 = None
    _debug = None
    _sag = None
    _stroke = None

    def __init__(self, sag, stroke, debug=False):
        self._sag = sag
        self._stroke = stroke
        self._debug = debug

    def _get_ideal_sag_mm(self):
        ideal = float(self._stroke) * (float(self._sag) / 100.0)
        if self._debug:
            utils.debug_print(self.__class__.__name__, '_get_ideal_sag_mm', ideal)
        return ideal

    def _get_ideal_pressure(self):
        slope, intercept = self._calculate_linear_equation()
        ideal = (slope * self._get_ideal_sag_mm()) + intercept
        if self._debug:
            utils.debug_print(self.__class__.__name__, '_get_ideal_pressure', ideal)
        return ideal

    def _calculate_linear_equation(self):
        x = np.array([self.measurement_100, self.measurement_150])
        y = np.array([100, 150])
        slope, intercept, __, __, __ = stats.linregress(x, y)
        if self._debug:
            utils.debug_print(self.__class__.__name__, '_calculate_equation', '{s}x +
             ↪  {i}'.format(s=slope, i=intercept))
        return slope, intercept

    def calculate(self):
        return self._get_ideal_pressure()
```

G.4 test_image_processor.py

```python
import warnings

from unittest2 import TestCase


class TestImageProcessor(TestCase):

    image_processor = None
    processed_test_image = None
    RS_IMAGE_PATH = '100_rs.jpg'
    FOX_IMAGE_PATH = '100_fox.jpg'

    def setUp(self):
        from image_processor import ImageProcessor
```

```
        self.image_processor = ImageProcessor('red', True)
        self.image_processor._image_path = self.RS_IMAGE_PATH
        warnings.filterwarnings('ignore')

    def tearDown(self):
        pass

    def test_show_image_normal(self):
        import cv2
        from image_processor import show_image
        image = cv2.imread(self.RS_IMAGE_PATH)
        result = show_image(image, 1)
        self.assertEqual(result, 0)

    def test_show_image_incorrect_file_path(self):
        import cv2
        from image_processor import show_image
        image = cv2.imread('')
        self.assertRaises(cv2.error, show_image(image, 1))

    def test_process_image_normal(self):
        import cv2
        image = cv2.imread(self.RS_IMAGE_PATH)
        gray_scaled = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
        blurred = cv2.GaussianBlur(gray_scaled, (5, 5), 0)
        self.processed_test_image = cv2.Canny(blurred, 0, 100, apertureSize=3)
        self.image_processor._image_path = self.RS_IMAGE_PATH
        processed_image = self.image_processor._edge_detect()
        self.assertEqual(processed_image.all(), self.processed_test_image.all())

    def test_get_ref_point_width_normal(self):
        self.image_processor._find_ref()
        self.assertIsInstance(self.image_processor._get_ref_point_width(), float)

    def test_get_ref_point_width_in_range(self):
        self.image_processor._find_ref()
        width = self.image_processor._get_ref_point_width()
        expected = 565.0
        self.assertTrue((expected*0.9) <= width <= (expected*1.1))

    def test_get_ref_point_width_not_none(self):
        self.image_processor._find_ref()
        self.assertIsNotNone(self.image_processor._get_ref_point_width(), float)

    def test_find_oring_black(self):
        self.image_processor._colour = 'black'
        self.image_processor._image_path = self.FOX_IMAGE_PATH
        self.image_processor._find_oring()
        self.assertIsNotNone(self.image_processor._oring)

    def test_get_measurement_px_normal(self):
        import numpy
        self.image_processor._edged_image = self.image_processor._edge_detect()
        self.image_processor._find_ref()
        self.image_processor._find_oring()
        self.assertIsInstance(self.image_processor._get_measurement_px(), numpy.int32)

    def test_get_measurement_px_in_range(self):
```

```
        self.image_processor._edged_image = self.image_processor._edge_detect()
        self.image_processor._find_ref()
        self.image_processor._find_oring()
        measurement = self.image_processor._get_measurement_px()
        expected = 1749
        self.assertTrue((expected * 0.95) <= measurement <= (expected * 1.05))

    def test_get_measurement_mm_normal(self):
        self.assertIsInstance(self.image_processor.get_measurement(self.RS_IMAGE_PATH),
        ↪   float)

    def test_get_measurement_mm_in_range(self):
        expected = 29.0
        measurement = self.image_processor.get_measurement(self.RS_IMAGE_PATH)
        self.assertTrue((expected * 0.9) <= measurement <= (expected * 1.1))
```

G.5 test_pressure_calculator.py

```
from unittest2 import TestCase


class TestPressureCalculator(TestCase):

    pressure_calculator = None

    def setUp(self):
        from pressure_calculator import PressureCalculator
        self.pressure_calculator = PressureCalculator(30, 57, True)

    def tearDown(self):
        pass

    def test_get_ideal_sag_mm_normal(self):
        self.assertIsInstance(self.pressure_calculator._get_ideal_sag_mm(), float)

    def test_get_ideal_pressure(self):
        self.pressure_calculator.measurement_100 = 29
        self.pressure_calculator.measurement_150 = 21
        self.assertIsInstance(self.pressure_calculator._get_ideal_pressure(), float)

    def test_calculate_linear_equation(self):
        self.pressure_calculator.measurement_100 = 29
        self.pressure_calculator.measurement_150 = 21
        self.assertIsInstance(self.pressure_calculator.calculate(), (float, float))
```

H IPO

I Week 9 Report

To do…

- ☐ 1 (p. i): Search for double spaces, punctuation
- ☐ 2 (p. i): Either upper case or lower case all PSI instances
- ☐ 3 (p. i): write acknowledgements
- ☐ 4 (p. ii): Write abstract
- ☐ 5 (p. 7): review section adding detail on setup method
- ☐ 6 (p. 14): complete hough transform section
- ☐ 7 (p. 15): What is going on here?
- ☐ 8 (p. 15): Add section or briefly explain
- ☐ 9 (p. 23): Remove statements saying this will be used
- ☐ 10 (p. 24): Could use
- ☐ 11 (p. 24): revise in light of dev methods
- ☐ 12 (p. 37): add source reference
- ☐ 13 (p. 41): adapt once statement comes in
- ☐ 14 (p. 41): Last name