University of Southern Queensland
Faculty of Engineering and Surveying

Machine vision and sensing with an Android

A dissertation submitted by

Mr Shaun Field

In fulfilment of the requirements of

Bachelor of Engineering (Electrical and Electronics)

September 2015

Abstract

This project investigated the ability for an Android mobile device to run an application that could automate a tractor. The development of such an application would lead to a cost effective, portable, and user friendly device that could easily be transported and installed on a tractor to allow vehicle automation. At the start of this project the method for automation had not been determined however the specific intent for the design of a machine vision application on an Android device was later defined.

The development of this application began with investigations into machine vision techniques and the Android SDK which identified the machine vision algorithm as well as the software libraries the application was be built upon. Access to the main video data was then achieved which enabled the manipulation of image data through accessing the pixel array information. Annotations were then added to the screen to allow for the output of data, and the line fitting algorithm selected for identifying crop rows was programmed. These achievements allowed the output of row identification and steering correction data to be added to the device screen.

These accomplishments concluded in an Android based machine vision application that is able to identify crop rows while processing the 30 fps 320x240 resolution image in an average of 34 ms per frame during typical running circumstances. This was done while keeping system RAM usage to an average of about 17 MB on a system that is also very tolerant to light fluctuations and noisy data.

Limitations of use

Candidate certification

I certify that the ideas, designs and experimental work, results, analysis and conclusions set out in this dissertation are entirely my own efforts, except where otherwise indicated and acknowledged.

I further certify that the work is original and has not been previously submitted for assessment in any other course or institution, except where specifically stated.

Shaun David Field

Student Number: 0061023577

Acknowledgements

This research was carried out under the supervision of Prof. John Billingsley.

This project leveraged BoofCV, the open source java computer vision library written and maintained by Peter Abeles.

Invaluable information and skills used for the project presentation were given by Mr Andreas Helwig, Dr Leslie Bowtell, Assoc Prof Tony Ahfock, and all other participants of the USQ ENG Project Conference 2015.

TABLE OF CONTENTS

List of figures

List of tables

Nomenclature and acronyms

The following abbreviations have been used throughout the text and bibliography:-

GPS	Global Positioning System
NCEA	National Centre for Engineering in Agriculture
USQ	University of Southern Queensland
SDK	Software Development Kit
RTK	Real Time Kinematic
RMS	Root Mean Square
XML	Extensible Mark-up Language
APK	Android Package
API	Application Programming Interface
RAM	Random Access Memory
OS	Operating System
App	Application
IDE	Integrated Development Environment

Chapter 1: Introduction

With the continually decreasing numbers of skilled farmers and the ever increasing necessity for agricultural production there is always a great need to increase productivity and efficiency on farms. Vehicle automation has been one of the key ways to do this in the recent past. This project investigates the concept of developing an automated vehicle guidance system using the Android mobile platform.

The Android platform has specifications for an array of sensors that can be used in vehicle guidance applications. This project researches the capability of these sensors being used in vehicle guidance applications with a particular focus made on machine vision techniques that use the camera as the main sensor for vehicle automation.

1.1 Outline of the study

This project aims to lower the cost and complexity of installing an automation system on a farming vehicle by creating a machine vision application on an easily installable and inexpensive mobile device. This is achieved by extending the work conducted by Billingsley (Billingsley & Schoenfisch 1997) and the National Centre for Engineering in Agriculture (NCEA) regarding tractor automation. A demonstrator program using the NCEA algorithm to deduce rows from an image was written and tested on an Android device and produced excellent results in the controlled laboratory environment. This research opens up the future possibility for field trials of an automated vehicle using an Android device. Further to this, other sensor data, such as GPS, accelerator, and geomagnetic, could be added to the program to try to improve the vehicle accuracy and response time.

1.2 Global food production

The United Nations Food and Agriculture Organisation expects worldwide food production will need to increase 70% by 2050 to sustain the predicted 9.1 billion people on the planet(FAO 2009). These figures indicate that there is a great need for increased agricultural production in the future. Despite the need for this increase in food production Australia recorded a drop of 5% of Australian farming businesses from 2011-12 with only 115,000 farming businesses recorded in 2012 (Australian Bureau of Statistics 2013). With decreasing farming businesses and the need for increased food production automation of farming vehicles is a logical solution.

1.3 Farm vehicle automation

Following rows of crops for long hours leads to operator fatigue and loss of concentration resulting in a decrease of precision (Rovira-Más et al. 2003). The automation of this task is shown to increase productivity and application accuracy resulting in greater crop yields while also enhancing operational safety (Ming et al. 2009). Global Positioning Systems (GPS) and machine vision are currently the two standard approaches for the automated control of agricultural vehicles used to follow rows of crops however other sensor based guidance methods are around (Emmi et al. 2014). .

1.4 Machine vision

Machine vision has been used for the automation of farming vehicles for a number of years now, however setting up vehicles for this type of automation requires specific technical skills and multiple pieces of expensive bulky equipment. This project extends the machine vision row following technique described by Billingsley (Billingsley & Schoenfisch 1997) by developing an Android application that shows the capability of replicating this machine vision technique on a small portable mobile device.

1.5 Android mobile operating system

Recent development in mobile technology has meant that inexpensive mobile devices have come on the market with inbuilt sensors similar to those used for automated vehicle guidance. This makes them a reasonable solution for implementing a tractor guidance system. Use of a mobile device would eliminate the installation of external cameras or internal processing units in tractor thereby reducing the cost and complexity of putting in automated guidance systems. Despite extensive research and development into tractor guidance systems, no research and development into implementing a tractor guidance system on a mobile device was found in the literary search for this dissertation.

1.6 Research objectives

This project aims to lower the cost and complexity of installing an automated guidance system on a farming vehicle by testing the capabilities of an Android device performing such a task using machine vision. The official project specifications can be found in 1.1.1.1.Appendix A along with changes that were made to the original specification. The updated project specification found in Appendix 1.1.1.1.A.2 identifies the research objectives as:

- **Review automated agricultural vehicle guidance systems**
  A review of agricultural vehicle guidance systems was conducted to identify the differing techniques and algorithms used in automated row following guidance.

- **Review the Android Software Development Kits (SDK) and Android sensors and devices**

A review of android sensors, devices, and software was conducted to identify available hardware and related software for items such as a camera that can be used for vehicle automation.

- **Write and test an Android demonstrator application**
  This objective involved accessing the video stream in Android memory, manipulating the pixel data, annotating the image, apply the NCEA line fitting algorithm, and outputting information to show steering correction data.

1.7 Conclusions

This project resulted in the development of a demonstrator application written on an Android device that is capable of identifying crop rows through the use of machine vision. This application has kept RAM usage to an average of about 17 MB while processing the 30 fps 320x240 resolution image in an average of 34 ms per frame during typical circumstances.

The application obtained image data through the device camera and processed the image using the NCEA machine vision algorithm. The processed data was able to accurately identify rows of crops and issue subsequent row following commands.

This demonstrator application displayed a regression line to the screen to visually identify the crop row as well as outputting numerical values representing steering corrections to be sent to the actuator. The application also included an adjustable threshold level to deal with fluctuations in changing lighting conditions.

Chapter 2: Farm vehicle automation

This chapter covers the history of automation for farming vehicles. An analyses of current navigation, computational, and control methods used in agricultural automated guidance systems is also conducted.

2.1 History

For over 60 years there has been research into the automated guidance of agricultural vehicles with automation to relieve the operator of continual steering adjustments the most frequently cited reason (Wilson 2000). Over these 60 years the operator of the vehicle has kept the same job of vehicle guidance and equipment operation however the vehicles have increase dramatically in size and power and this increase in vehicle power has led to an increase in vehicle speed and equipment size. This increase in speed and size has made the operators job of staying on course all the more valuable and difficult as any deviation will result in a greater area missed or double worked.

Having a vehicle that stays on course is not only economically viable but the decrease use of chemicals, fuel consumption, and improper soil tillage make it environmentally viable too. With these things in mind the need to automate vehicle guidance is ever more desirable. Autonomous guidance of a vehicle not only enhances operator safety, but it also increases accuracy and productivity resulting in better economic return and better environmental impacts (Han et al. 2004).

Automated vehicle guidance has been implemented in many fashions over the last 60 years. Morgan (Morgan 1958) and later Brooke (Brooke 1972) wrote about a tractor automated by buried leader cables. Palmer and Matherson (Palmer & Matheson 1988) and also Searcy (Searcy et al. 1990) wrote about the use of radio beacons positioned in a field as a method of automatically navigating a vehicle. These two early autonomous methods were not implemented in many agriculture situations due to the initial expense of the equipment and the limited capability of the radio beacons. It wasn’t until the 1980’s with the exploration of machine vision that farm vehicle automation began to become reasonably priced with respectable accuracy as described by both Reid (Reid, Searcy & Babowicz 1985)and Gerrish (Gerrish et al. 1985). Reports by Larsen (Larsen, Nielsen & Tyler 1994) and Bell (Bell 2000) then note the use of GPS based guidance systems emerging in the 1990’s.

Other techniques such as optical guidance, mechanical guidance, and ultrasonic guidance have also been printed by Reid (Reid et al. 2000) and Tillet (Tillett 1991), however the two leading technologies in autonomous navigation are GPS followed by machine vision. Today fused GPS and machine vision systems are becoming more popular as each of these systems have their advantages and disadvantages and they complement each other quite well (Ming et al. 2009). Whatever the technique is that is used, Emmi (2014) separates all modern automation of farming vehicles into to three main modules:

1. **Sensing** – This is the collection of information from the surrounding environment.
2. **Decision making** – The sensors lead to a piece of hardware that uses a software algorithm to determine the best course of action for the vehicle to take.
3. **Acting** – After decision making the hardware/software device sends a message to the vehicle to carry out the task it deemed most appropriate.

2.2 Machine vision

Machine vision is one of the main techniques used for the automated guidance of agriculture vehicles. Machine vision is a relative control mechanism as it relies on a camera image to tell the vehicle its current position relative to the image captured by the camera. While there are varying techniques used in vehicle guidance machine vision applications, all machine vision for agricultural vehicle automation can be broken down into three main areas.

1. **Image acquisition** - An image is taken of the vehicle surroundings, it is then digitised and placed in memory. This is generally done through a camera mounted on the farming vehicle.
2. **Image processing** – The image is then placed through an algorithm that identifies the vehicle’s location and then outputs control signals to the vehicle’s actuator.
3. **Output control** – The signal controls are received by an actuator that is used to steer the vehicle in the desired direction.

An advantage of using machine vision is that no pre-programmed set of co-ordinates to guide a vehicle are needed making it useful for terrain that has not been mapped out in the past. Another advantage of machine vision the that it is relatively insensitive to surrounding landscape conditions which allows machine vision to work in covered areas where other machine guidance techniques such as GPS will not work well.

Machine vision does however have the disadvantage of needing more sophisticated computational techniques and it is also possible for these techniques to loose accuracy if there is some form of image distortion such as shadows, weeds, or missing crops.

In his 1997 paper Billingsley (Billingsley & Schoenfisch 1997) explains a technique used by the NCEA which acquires a colour image in YUV format from a camera mounted on the tractor. The tractor operator then selects a viewport window that straddles a crop rows. The pixels within this viewport are then analysed using a threshold level to identify the greenness of each pixel. A linear regression algorithm is then applied which returns a fit and a slope of the regression line. As the plant will appear mainly in the centre of the window, the regression line will give a path for the tractor to follow. This technique displayed an accuracy of 2cm at approximately 6.9 m/s.

Shen (Shen & Liu 2007) used a similar technique where an RGB image from the camera doubles the green to make a R2GB image. Each pixel is then checked against a threshold. This image is then segmented into greyscale where it is again compared against a threshold. If the threshold is met the pixel is then changed to white otherwise it is changed to black. The image is then processed using dilation processing, an acnode filtering technique, a midpoint encoder operation, and is finally improved with a Hough transform. This machine vision technique allowed row following at 3.5 m/s.

Jiang (Jiang, Wang & Liu 2015) recently reported on a technique that uses a five stage analysis technique consisting of methods similar to Shen. The camera image is transferred into grey scale and it is then checked against a threshold and transferred again into a black and white image. An estimation of the row centre points is calculated and multiple regions of interest are found. A clustering method is then used to confirm the centre points of the rows. This is finally followed by a linear regression technique to determine the centre for the crop rows. This technique reports accuracy and processing time greater than a standard Hough transform method.

A study carried out by Zhang (Zhang, Cheng & Zhang 2008) also used a R2GB image converted to grey scale and finally to back and white using a threshold. A horizontal scan of the image pixels then identifies the target regions and points. These target points are then clustered according to the abscissa of two adjacent scanned lines. Three clusters are passed through a known point Hough transform to identify a regression line and the crop rows.

2.3 GPS

GPS.Gov ('GPS.Gov' 2015) describes GPS as a United States Department of Defence owned collection of 24 geo-synchronous space satellites that broadcast location and time information to any position on earth where there is line of sight between the satellite and the receiver. GPS.Gov describes GPS as being essential for the development and implementation of precision agriculture. While GPS was the first satellite system to broadcast location and time information other countries around the world now have their own satellite systems in place such as the Russian GLONASS and the European GALILEO satellites.

The GPS Standard Positioning Service Performance Standard (Defense" 2008) Specifies that the worst case for accuracy of the GPS service is a pseudorange accuracy of 7.8 meters at a 95% confidence level, which equates to a worse case of about 3.5 meters horizontal accuracy. The horizontal accuracy is effected by environmental conditions between the GPS receiver and the GPS satellites. Clearer environmental conditions and access to a greater number of satellites will improve the horizontal accuracy of the GPS receiver.

In the 1990’s GPS started to emerge as a way to automate the control of an agricultural vehicles (Ming et al. 2009). Unlike machine vision, GPS gives the absolute position of the automated vehicle and therefore must map out the GPS co-ordinates for the vehicle to follow. These co-ordinates are generally saved when the crops are planted. Alternatively, the co-ordinates can be programmed into the GPS so that the automated vehicle will know which path to follow. Both Abidine (Abidine et al. 2002) and Gan-Mor (Gan-Mor & Clark 2001) mention the versatility and accuracy of GPS when used for agriculture tasks such as sowing, tilling, planting, cultivating, weed control, and harvesting. Because of its accuracy, ease of use, and its ability to not be affected by any inconsistency in visual camera data it has become the most popular technique in automated guidance of agricultural vehicles.

While an accuracy of 3.5 meters isn't precise enough for use in vehicle automation, various signal correction techniques can be implemented to correct the inaccuracies of GPS making it accurate enough for vehicle automation. Keskin (Keskın, Say & Görücü Keskin 2009) evaluated low cost GPS receivers for precision in agriculture and found an RMS error of 1.48m on straight crop row tests. Gan-Mor (Gan-Mor, Clark & Upchurch 2007) notes that differential correction systems are often put in place to reach accuracies below 1 meter. Gan-Mor reports that a differential correction system called Real Time Kinematic (RTK) GPS is used to gain an accuracy down to about 1cm which has made this technique very popular for automatic guidance systems in row-crop operations. There are a great number of examples of the accuracy obtained with RTK GPS two of which are Sun (Sun et al. 2010) who recorded a 2cm RMS error on row crops and Perez-Ruiz (Pérez-Ruiz et al. 2012) who reported an RMS error of only 0.8cm when using RTK GPS on crop rows. Kise (Kise et al. 2002) even obtained RMS errors of only 6cm when using RTK GPS on a sinusoidal path.

As with any differential correction system RTK requires the use of a base station and complicated algorithms to reduce or remove any errors between the base station and the GPS receiver. This added complexity dramatically increases the price of the equipment and is one of the downsides to RTK GPS (Gan-Mor, Clark & Upchurch 2007). Another disadvantage of GPS is that the signal will not work in shielded areas and will loose accuracy if the right environmental conditions aren't met.

2.4 Other sensing techniques

There are various other sensing techniques that have been mentioned in several studies however none of these techniques currently have the popularity that GPS and machine vision currently hold. Odometry was used in experiments by Borenstein (Borenstein 1998) and later by Perez-Ruiz (Pérez-Ruíz et al. 2014) however odometry tends to accumulate errors rather quickly eventually leading to large lateral errors in the vehicles location.

Subramanian (Subramanian, Burks & Arroyo 2006) and Noguchi (Noguchi et al. 2002) both noted the use of laser-based sensors in conjunction with GPS and achieved errors of less than 2.5 cm and 1 cm respectively. Laser-based sensors setup reflectors around the field and triangulate the vehicles position by bouncing lasers off of these reflectors. This technique has the advantage of working in different types of environmental conditions however it has shown some faults due to laser measurement distortion when the vehicle is traveling on uneven ground.

Ming (Ming et al. 2009) mentions both accelerometers and geomagnetic sensors that have also been used as sensing techniques. Accelerometer use alone was prone to positional drifts however its use has shown good results when combined with other sensing techniques such as in Noguchi's (Noguchi & Terao 1997) experiment which resulted in less than 5cm error when combined with RTK GPS. Geomagnetic sensors also worked better when combined with other sensors as noted by Benson (Benson et al. 1998) when he combined a geomagnetic sensor with a medium accuracy GPS to achieve an error of less than 1cm.

Chapter 3: Android Software Development Kit

To date sensors and decision making hardware used for automated guidance have involved bulky equipment that must be fixed to agricultural vehicles, however modern mobile devices have GPS and camera systems built in as well as other sensors that can be used for automated guidance. Of the many mobile devices available the Android operating system has a significant share of the market and supports software and hardware for GPS, video camera, accelerometer, gyroscope, light sensor, Bluetooth and other sensors useful to build an automated guidance system (Moore et al. 2014)

This section of the dissertation covers the specifics about the Android Software Development Kit (SDK) that was used for development of this project. This section omits the hardware specifications for Android devices as each device has differing hardware options dependant on manufacturer and model. The underlying software for control of this hardware however is covered as all android device software is run using the Android SDK. All specifications in this section of the dissertation are taken from Android's official website ('Android Developers' 2015).

While this chapter gives a general overview of the Android SDK, Chapter 5: has the specific details of how Android software was used during the creation of this project

3.1 Background

The Android platform involves Android applications that are installed and run on devices that use the Android operating system. The Android operating system (OS) is the world's most popular mobile operating system and is deployed on hundreds of millions of devices in over 190 different countries around the world. The OS is built using open-source Linux and controls all of the software and hardware within the mobile device. Android version 5.1 is the most recent operating system which introduced support for 64-bit architectures.

Android applications are written in the Java programming language and makes use of Extensible Markup Language (XML) resource files. The Java files tell the Android OS what the program is to do while the XML files tell the operating system what resources are needed to run the application. When compiled the Android SDK will package all of the Java code and XML resource files into an Android package (APK). This APK file is then installed on Android devices within its own virtual machine and it is within this virtual machine that the application is run. This allows the OS to control what hardware and software each application has rights to access.

The android SDK is broken into software packages that control different portions of the android device. This project dealt mainly with accessing the Android sensors and Android memory. All hardware devices are accessed using the android.hardware software package of the Android SDK and all Android devices use Android Random Access Memory (RAM) for computation and decision making. The Android OS virtual machine performs routine garbage collection to free memory that is no longer in use.

3.2 Camera

The camera is the main sensor used for this project. It is a hardware based sensor that has different specifications for every Android device and is controlled using the android.hardware.camera class for all Android software prior to Android 5.0, and the android.hardware.camera2 class for all devices running Android 5.0 or higher. Android 5.0 was released in November 2014. The android.view.SurfaceView class is another useful camera class that is used within this project to present a live camera previews frames both to other sections of code and to the device screen. Additional details of how the camera was used during this project can be found in section 5.4

3.3 GPS

Android GPS accesses satellite data to specify the device's current longitude and latitude in degrees, minutes, and seconds. The software used to access this sensor is held in the android.locations class. GPS differential correction techniques cannot currently be used on Android devices without additional expensive hardware which restricts the accuracy of Android GPS to 3m. This inaccuracy without additional hardware made Android GPS not acceptable use in this project.

3.4 Other sensors

3.4.1 Accelerometer

An accelerometer is a hardware based sensor that has different specifications for every Android device and is used to measures the acceleration force, in m/s^2 , that is applied to all three physical axis (x,y,z). Almost every Android device has an accelerometer and they use about 10 times less power than other motion sensors. Common uses for an accelerometer are to detect the motion of an object in a given direction. The x, y, and z motion values will be 0 m/s^2 when the device is stationary, will increase when moved toward the arrow, and will decrease when moved away from the arrow. All axis are also subject to the force of gravity ($9.81 m/s$). A high pass filter can be applied to the accelerometer to remove the force of gravity and give linear acceleration only, alternatively a low pass filter can be applied to the accelerometer which isolates the force of gravity.

The accelerometer data is held in the android.hardware class. An instance of Sensor.TYPE_LINEAR_ACCELERATION must be created to access the linear acceleration accelerometer data. While accelerometer information has not been used in this dissertation, details of the Android accelerometer have been included as it does have the potential to be used in the future development of the Android vehicle guidance method that has been used in this dissertation.

3.4.2 Geomagnetic

The Android geomagnetic sensor is a hardware based sensor. The geomagnetic sensor software data is held in the android.hardware class and is used to measures geomagnetic force in μT on the x, y, and z axis. As with the accelerometer information, geomagnetic information has not been used in this dissertation yet details of the Android geomagnetic sensor have been included as it does have the potential to be used in the future development of this Android vehicle guidance method.

3.5 Data handling

Applications run on an Android device have access to system RAM, data storage, and data processing capabilities. Each specific device will differ in hardware specifications but each will use the same Android SDK classes.

Android uses an automated garbage collector to free up RAM resources that are no longer needed and also uses both paging and memory mapping to manage memory. Any created object will remain within RAM until the app releases the object for collection by the garbage collector, which will run more often when there are more resources in RAM. Android recommends that background services are used sparingly and terminated when no longer in use to help free system RAM.

Android devices are increasingly being released with multiple processors and the Android SDK version 3.0 and above makes it possible to run threads in parallel. This is achieved through the use of the Java Runnable and Thread classes in conjunction with the ThreadPoolExecutor class. Running different threads on different processors will allow for parallel processing of data as is the case with the application built during this project.

While the developed application does not store any data permanently, the main Android data storage options are still shown here for reference.

- Key-value sets – This is used to save a small collection of key-values. It uses the SharedPreferences API where creation of a SharedPreferences object creates a file containing key-value pairs and provides a method to read and write them. The created file can be either private or shared and is managed by the framework.
- A File – This is used to store large amounts of data that are intended to be read start to finish. A file can be stored on the device's internal or external memory. Internal memory is used to ensure that the user nor other apps can access the file. External memory is used for any file that doesn't require access restrictions and for files that can be accessed with a computer.

- SQL Database – This is used to store structured data that can be read in any order. The android.database.sqlite is the Android package that is used. This technique requires a database schema and can be stored on internal or external memory. Access restrictions can also be applied.

3.6 Bluetooth and Wi-Fi

Wi-Fi and Bluetooth are two of the sensors that Android include in their SDK that theoretically can be used to output information from an Android device to the automated vehicle actuator controller. The use of these sensors to output control information is beyond the scope of this dissertation and will be left for the future development of this Android vehicle guidance method

Chapter 4: NCEA vision guidance system

The machine vision technique used for row following that Billingsley (Billingsley & Schoenfisch 1997) noted in the NCEA study in 1997 has been adopted as the machine vision algorithm for this project. This algorithm was selected for its simplicity and ease of portability to an Android platform. This chapter covers the specifics of this NCEA machine vision guidance system.

4.1 History

The NCEA began research into machine vision use in agriculture in the early nineties. Since that time the NCEA has developed a machine vision animal identification system, a macadamia nut counting system, and a vehicle automated guidance system.

The development of a machine vision guidance system lasted three years and resulted in a machine vision prototype that was relatively insensitive to weeds and could withstand the fading in and out of crop rows while still keeping the vehicle on the correct bearing. This three year study involved cameras being externally mounted onto the tractor, as well as internal processing units being integrated into the tractors. Six prototypes of this machine vision technique were tested in the field and results showed that an accuracy of 2 cm was able to be maintained at 35 km/h.

This system originally used a black and white camera taped to the roof of a David Brown tractor. The captured image was 768x96 pixels and was transferred over cabling to a PC installed inside the tractor by direct memory access where a program written in C processed the image data. This program then output steering commands over a cable to a stepper motor connected via belt drive to the steering wheel of the tractor. If there was some type of system error an audible warning sound was outputted through the PC speakers and the system would revert to manual control. Although there have been several refinements to this system as technology and resources became available, the basic flow of information and the connection of hardware elements has not changed much between versions.

During field tests the overall response from users was positive with all users impressed with the speed, accuracy and efficiency of the system. Complaints that were received from the users were regarding the complexity of setting up and calibrating the system. These complexity issues may be fixed with further development of the Android application developed in this paper.

4.2 Image acquisition

The original camera was mounted externally on the tractor's roof and obtained a black and white 768x96 pixel image but the latest version used a miniature camera connected to the bonnet of the tractor to obtain the 643x480 YUV image used for processing. The change to a YUV image allowed the chrominance and the intensity of each pixel to be held in separate data bits which makes it less sensitive to changes in lighting conditions.

The image is then moved into memory using direct memory access where each pixel is compared to a threshold level that is set at the start of the program. The original system would degrade system performance by switching the video output between the processor and a connected monitor. Later models allowed dual image streams so that the video feed could be viewed and annotated with real time data without reducing overall system performance.

The tractor operator assigns adjustable viewports to the incoming image which identify the pixel locations of one row of crops. Then each pixel within each of these viewports is compared against the threshold level set at the start of operation. Pixels that are greater than this threshold level are marked as plant and are used for further processing.

The tractor operator also sets the proportion parameter at the start of the program which indicates the expected proportion of pixels identified as plant within each window. This proportion value depends on the growth stage of the plant with typical values being 0.1 for a new plant to 0.5 to a mature plant. The total number of plant pixels for each window in each frame are then compared and this data is used to increment or decrement the next frames threshold level. This makes the system very insensitive to fluctuations in lighting conditions.

4.3 Image processing (Identification of rows)

The acquired image data is processed using a technique similar to linear regression to identify the centre of each crop row. As each positioned viewport contains only one crop row, drawing a line which minimises the viewport moment of inertia when spun around this line identifies the line of best fit and the centre of a crop row. Billingsley (Billingsley & Schoenfisch 1997) showed the cost function to calculate the moment of inertia for each viewport as:

$$\mathrm{C} = \sum_{\mathrm{x.y_e window}} \sum \mathrm{m(x,y)} * (\mathrm{x} - \mathrm{offset} - \mathrm{slope} * \mathrm{y})^2 \qquad (4.1)$$

where: x is the x coordinate

y is the y coordinate

$window$ is perimeter of the viewport window

$m(x,y)$ is a matrix of x and y coordinates identified as plant

$offset$ is the x coordinate for the line of best fit

$slope$ is the slope of the line of best fit

Minimising this cost function minimises the total error of all of the data points within the viewport. This is achieved by identifying the offset and slope values for the line of best fit. The line of best fit slope and offset parameters were calculated within the C program by providing solutions to the following simultaneous equations:

$$\frac{\partial C}{\partial \text{offset}} = 0 \text{ and } \frac{\partial C}{\partial \text{slope}} = 0 \qquad (4.2)$$

Which Billingsley identified can be solved using the following formulas for identifying the lateral offset corrections and the slope corrections respectively:

$$fitOffset = \frac{mx*myy - mxy*my}{m*myy - my^2} \qquad (4.3)$$

$$fitSlope = \frac{m*mxy - mx*my}{m*myy - my^2} \qquad (4.4)$$

where: m is the total number of pixels identified as plant
mx is the total x axis moment about the viewport horizontal line
my is the total y axis moment about the viewport vertical line
mxx is the second moment of area
myy is the second moment of area
mxy is the total second moment

After the offset and slope values for the line of best fit are identified, the correction values needed to obtain these offset and slope values are calculated. These correction values are then converted into steering commands outputted to the actuator that steers the tractor.

A ratio to check the quality of the results is run by comparing the moment of inertia about the line of best fit against the moment of inertia about the horizontal axis. This ratio gives a “quality” value and the correction data is only acted upon if the quality is greater than 4. This increases the accuracy of the program by only acting upon good quality data containing adequate plant values and not acting upon data with scattered plant values due to weed growth, pests eating the crop.

While only one viewport is needed for vehicle automation the use of a a greater number of viewports is recommended so that vehicle automation can continue in the event of a viewport not being able to output correction data to the actuator. If all viewports contain poor quality data 3 times in a row, then an alarm sounds and the vehicle control reverts back to manual.

A simplified diagram of this crop row detection algorithm is shown below in Figure 4-1.

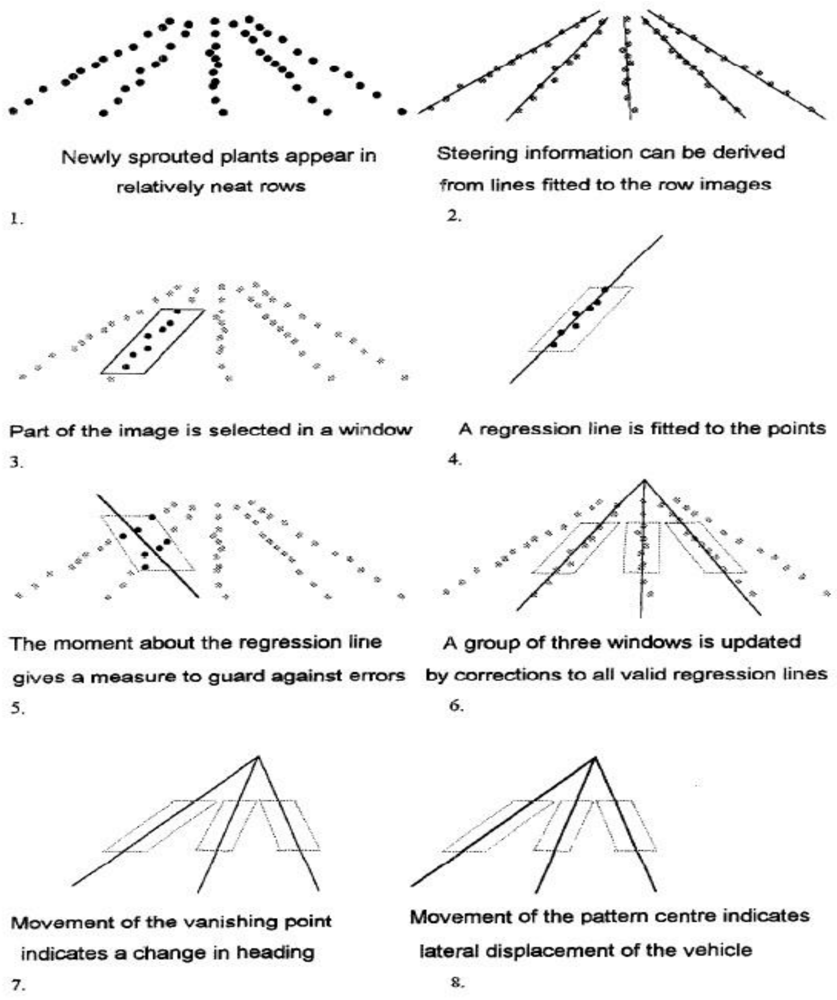

Newly sprouted plants appear in relatively neat rows
1.
Steering information can be derived from lines fitted to the row images
2.
Part of the image is selected in a window
3.
A regression line is fitted to the points
4.
The moment about the regression line gives a measure to guard against errors
5.
A group of three windows is updated by corrections to all valid regression lines
6.
Movement of the vanishing point indicates a change in heading
7.
Movement of the pattern centre indicates lateral displacement of the vehicle
8.

Figure 4-1: NCEA linear regression image analysis algorithm (Billingsley & Schoenfisch 1997)

4.4 Actuator control

The correction values produced after the algorithm has been run are converted into steering commands outputted to the actuator that steers the tractor. The lateral movement steering commands are calculated using either the fitOffset correction value, or by identifying the "vanishing point" of where the best fit lines from multiple viewport meet at a point. The fitSlope correction value is used to identify the angular displacement of the vehicle and issues steering commands accordingly. These are just the basics for the actuator control and further detail has been omitted as it is beyond the scope of this project.

4.5 System testing and evaluation

Two main tests were conducted under controlled conditions to identify the systems capabilities. The first test was carried out to evaluate the performance of the machine vision algorithm. In a laboratory the camera was pointed toward a piece of white paper connected to a stepper motor and the ability of the machine vision algorithm to track the movement of the paper from left to right was determined. The actual location of the paper and the detected position of the paper were recorded independent of each other and locations of the paper were then compared. Figure 4-2 below shows that the algorithm achieves good quality results. This figure shows the actual position of the paper as the top line of dots and the detected position of the paper as the bottom line of dots.

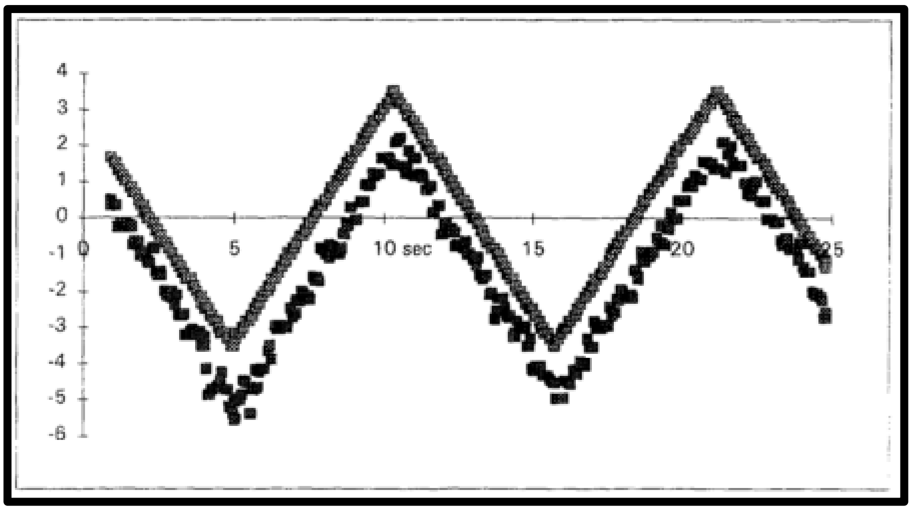


Figure 4-2: Actual vs captured data testing the performance of the machine vision algorithm (Billingsley & Schoenfisch 1997)

A full system test was then carried out where white lines were marked on the ground and the capabilities of the whole system were evaluated. During this test a secondary camera was attached to the axil and recorded the system performance. The data for this test was taken back to a laboratory and results were recorded. These results are shown below in Figure 4-3 and they show that the system has achieved an accuracy of 2cm.

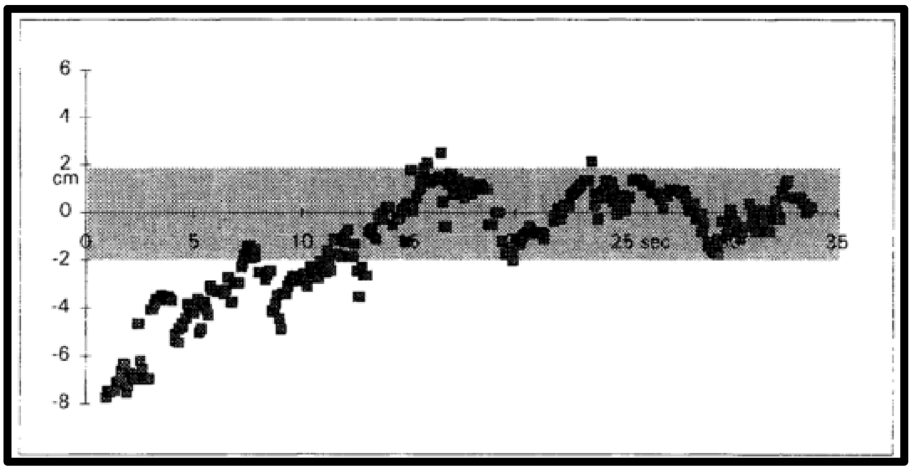


Figure 4-3: Results from the NCEA full system test at 1 m/s.

Chapter 5: Methodology

This chapter identifies the method used to develop an Android based autonomous farm vehicle. This involved writing several short programs in Android code to identify some key aspects of the Android SDK followed by writing more specific code used in this projects' Machine Vision demonstration application. The machine vision algorithm discussed in Chapter 4: was then applied on an Android device before code optimisation was carried out

5.1 Development tools and techniques

This section covers the programming methodology used during this project and lists the Integrated Development Environments (IDEs) used for development and testing of this code.

5.1.1 Agile programming technique

The agile programming techniques described by Martin (Martin 2003) was used for the development of software during this project. The agile programming technique generates small portions of computer code before testing its accuracy and usefulness. This allows for a lot of usable code to be written and tested in a short time as not a lot of preparation goes into the planning process for the long term goals of the program. As Dingsoyr identifies (Dingsoyr 2010) that with agile programming there is a vague idea of what the end goal will be however more focus is made on individual components that need to be developed now in order to get useable software as quickly as possible. This leads to a very fast cycle of planning, requirements analysis, design, coding, unit testing, and acceptance testing.

An agile programming methodology was selected over the more traditional waterfall method for the following reasons the following reasons as stated by Martin (Martin 2003):

- Risk management – This project involved a lot of risk as a lot of aspects of its development were unknown at the start of the project and as states, Agile programming has the ability to minimise this risk because of its ability to adapt to change so quickly. Small unit testing of code quickly defines the usefulness of this code thereby eliminating the possibility of writing a lot of code only to find in a few weeks' time that it isn't useful for this application.
- Development speed – Martin (Martin 2003) also states that agile programming allows a lot of working code to be written in a very short amount of time which was needed in the development of this project due to project deadlines.

Dingsoyr (Dingsoyr 2010) also specifies the following three reasons for using the agile coding method:

- Quality – Code is tested as individual components are created so the quality of each individual segment is guaranteed prior to full system integration.
- Design – A test driven approach is used to define the final requirements of the system allowing the system to be designed and refined as system modules are built.
- Segmentation – Agile programming segments code so only one aspect is studied at a time making it suitable for this project.

While agile programming has many benefits suitable for this project many published authors such as Rierson (Rierson 2013) do not recommend agile programming for safety-critical software projects such as those used in the automotive industry. The Institute of Electrical and Electronic Engineers' ('IEEE Standard Glossary of Software Engineering Terminology' 1990) definition of a safety-critical software is any software where the failure can lead to a hazardous state, so by definition the automation of any vehicle falls under this category. The International Organization for Standardization's (ISO) international standard ISO26262 for road vehicle functional safety defines a coding requirement for road vehicle functional safety that is more structure based like that of the waterfall model.

Although this project will eventually lead to a safety-critical software project, at this stage of development there are no safety-critical aspects involved. All testing for this project has been run as simulations in a controlled laboratory therefore allowing the use of the agile programming method. Future development of this application would have to review the programming methodology used and then take that into consideration when designing the enhanced system.

While the benefits of using Agile programming for this project far outweigh limitations a few things that were considered during the development of this project were:

- System Testing - During agile programming components are assumed to interact nicely if they work as individuals but this is not always the case. Agile programming doesn't test with a system as a whole until the final stages of development.
- Abstract code – Agile programming can lead to a lot of abstract code which Android Developers ('Android Developers' 2015) list is not a desireable thing when coding for a mobile because of the memory usage.

5.1.2 Integrated Development Environment (IDE)

Android Developers ('Android Developers' 2015) recommended Android Studio IDE Version 1.3.1 was used to develop and test all of the elementary code found in 1.1.1.1.Appendix B, however after some external computer vision libraries started to be investigated some compatibility issues became prevalent between the computer vision libraries, Android Studio, and the Java Development Kit (JDK) version 1.7 that was being used. After these issues could not be resolved the Intellij IDE was used for the remainder of the coding and testing.

5.2 Elementary code

This section covers elementary code developed to become familiar with the Android SDK and the recommended best coding practices to follow during development. Some available computer vision libraries are also investigated in this section.

5.2.1 Android developers

Initial coding design and development for all basic Android modules followed the online training modules defined by Android developers ('Android Developers' 2015). Android developers best coding practices. Android Developers also has a section for the best coding practices to follow during program development. As this project was completed using the agile programming method which primarily focuses on writing working code before optimising code, the first stages of this project didn't follow all of these coding practices however after the correct working code was found it was then cleaned up and altered to incorporate some of these Android coding practices regarding memory management, and Multithreading. One limitation of this application is that some of the coding practices used are not optimal such as leaving application fatal operations outside of a try, catch block. This was due to deadlines for project completion. Details about the Android development environment learnt from doing this developer training can be found in 1.1.1.1.Appendix B.

5.2.2 Computer Vision libraries

Two open source computer vision libraries were investigated to see computer vision techniques that are available. The two libraries investigated were OpenCV and BoofCV.

- **OpenCV** – a C++ based computer vision library

OpenCV.org (OpenCV 2015) states that this bundle of computer vision classes and libraries is available under a BSD license making it free for both academic and commercial use. Development began in 1999 and while OpenCV is primarily a C++ programming language it also has Java interfaces and it supports the Android operating system.

- **BoofCV** - a Java based computer vision library

Abeles (Abeles 2012) records that this computer vision package of classes and libraries has been released under the Apache 2.0 license making it freely available for both academic and commercial use. Development of BoofCV began in 2011 and it is written in Java making the wide range of prewritten libraries and example code compatible with the Android operating system.

Example code segments using both OpenCV and BoofCV were reviewed with both libraries being suitable for this project. While OpenCV (OpenCV 2015) claims that their open source library is the fastest because it is written in the native C language Abeles (Abeles 2012) claims that BoofCV is faster when processing higher level algorithms. Neither of this data could be tested and confirmed.

In the end BoofCV was selected over OpenCV as the primary computer vision library for use in this project primarily because the classes are written in Java which is compatible with Android systems with only a slight bit of modification. Due to time constraints and a slight pre-existing developer knowledge in Java BoofCV was the best option available.

5.3 Android machine vision program development

The following section covers the entire development process used to create an Android based application that is successful in identifying crop rows. The Android package files generated during this process can all be found in 1.1.1.1.Appendix C.

5.3.1 Program description

The main aim of this application is to test if a machine vision program capable of identifying crop rows can be successfully developed and run on an android mobile device. The algorithm detailed in Chapter 4: that identifies straight crop rows was selected for application development and all testing of this application was conducted in a controlled environment under controlled conditions.

The program begins by obtaining an image of the upcoming crop rows from the Android devices' rear facing camera. The image is then converted into a workable format before a viewport is set around one row of crops. Each pixel within the viewport is then compared against a threshold level used to identify plant and non-plant pixels. The threshold level is then adjusted by finding the proportion of plant pixels within the viewport and checking it against a predefined proportion setting. Then the plant pixels are placed through an algorithm similar to the NCEA algorithm defined in section 4.3 of this document. If the quality of pixels within the view window is good, then a regression line and correction data to be passed to the steering actuator are annotated onto the video frame image before it is output to the device screen for display. If a bad quality fit is found for three successive frames then the annotated regression line is not displayed on the screen simulating the system reverting back to manual control. Implementation of the steering actuator itself is outside the scope of this project and as such the application process finishes when steering correction information is annotated to the screen. A flow diagram showing the flow of information for this process is found in Figure 5-1. This flow of information repeats for every frame that the camera produces.

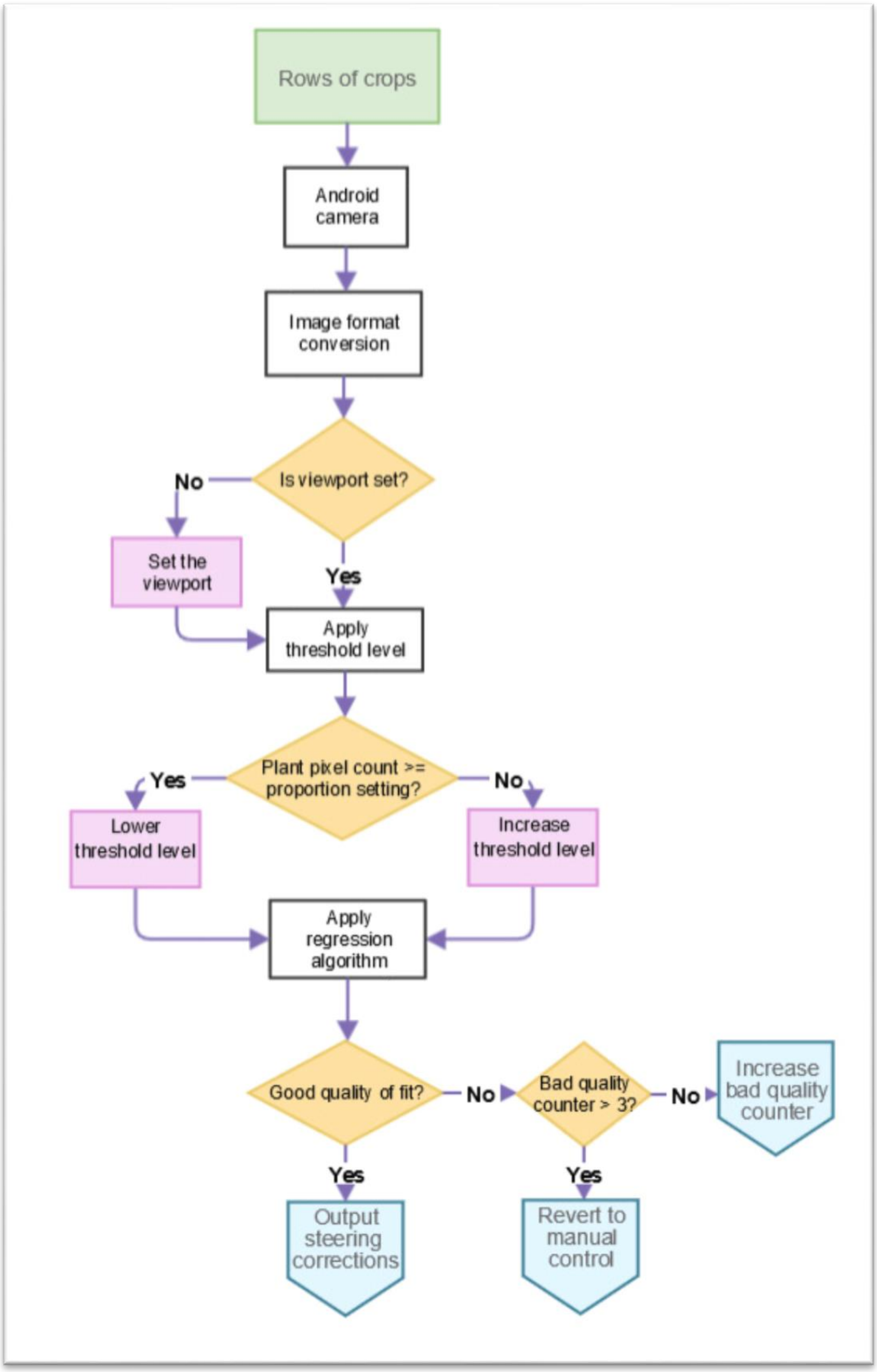


Figure 5-1: Application flow diagram

5.3.2 Application limitations

This application has the following limitations

- The algorithm is designed to identify straight crop rows and as such doesn't have the ability to identify crop rows that are curved.
- This is an elementary test version of an Android based machine vision application and not a production version. One result of this elementary code is that there is no user interface for this application and any changes in system constants has to be hard codded and recompiled in order to work.
- Testing in a controlled laboratory ensured perfect conditions for the machine vision application to function. Testing with noisy data was beyond the scope of this project therefore the applications ability to work in an uncontrolled environment is unknown.
- During testing an error of half a pixel length was identified and the source of the error could not be located before this document was published. Details of this error can be found in section 6.5 .
- This application does show the vanishing point information with what is assumed to be the correct calculations for identifying this point, however testing for the accuracy of this value had not been performed in time for this document to be written.
- This application does not follow all of the recommended Android development coding practices and as such some of the code developed is not optimised for performance or usability. Such things as leaving application fatal operations outside of a try, catch block have not been followed which can lead to the application terminating when certain system parameters aren't met. Due to deadlines for project completion the code could not include all recommended practices.

5.3.3 Initial development setup

After the initial investigation into Android coding and available computer vision libraries the development parameters in Table 5-1 were put into place.

Table 5-1: Initial Android development parameters

Parameter	**Value**	**Reason**
IDE	Intellij	Intellij was selected over Android Studio due to some compatibility issues between Android Studio, the latest JDK and the BoofCV libraries. This issue would cause the system to crash on start up when the program was run or debugged on an Android device.
JDK	7	JDK 7 was selected to keep in line with the latest BoofCV library compilation versions.

Computer Vision Library	BoofCV	BoofCV was selected because the libraries are written in Java which makes them compatible with the Android Operating System.
Android SDK	Min – 10 Target – 17	Android version 10 (2011) is the minimum version that BoofCV has been fully tested and Android version 17 (2013) was the version loaded on the equipment used for testing.

The AndroidManifest.xml file specifying the Android SDK and the application start-up values can be found in 1.1.1.1.Appendix C.

The application starts by accessing the Row_Follow_Main.java class as specified by the Android manifest. This class begins by importing all of the Android, Java, and BoofCV libraries needed for operation. This main class extends the Android Activity.java class allowing the application to display data to the screen and to securely deal with interruptions to the application, and it also implements the PreviewCallback.java class which delivers copies of the video frames as they are captured by the camera. The global constants and variables used by the application are then specified.

The onCreate method is the first method that is called which requests access to the main screen from the Android operating system and then creates the CameraPreview and Visualisation objects before adding them to a FrameLayout object that allows these two items to be output to the device screen. The created Visualisation object uses its onDraw() method to scale the Bitmap image stored in the 'output' variable to the required size and then attaches it to the canvas for output to the device screen. The 'output' variable is a copy of the camera image converted into Bitmap format. This variable is updated frame by frame as new images arrive from the device camera. This variable is accessed by two separate threads so write access to it needs to be synchronised to avoid information mismatch. The CameraPreview.java class and the use of threads is explained in detail in section 5.4 1.1.1.1.Appendix B also contains information regarding threads.

Additional methods such as onPause() and onResume() which handle interruptions to the application are available in the event device applications are switched.

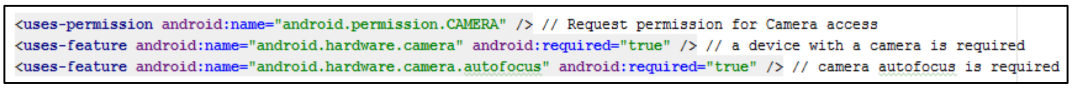
The last method of interest in the initial setup of the application is the setUpApp() method. This method is responsible for assigning initial values to global variables. It sets up the camera and data output options and connections, the annotation and viewport styles and sizes are also defined here, and the second processing thread is created and started within this method. Each of these setup parameters are covered in the subsequent sections within this chapter.

5.4 Video stream access

This application gets a 320x240 image at 30 fps in NV21 format from the device camera which is then converted into RGB format. Calculations for plant identification and for further use in the row following algorithm use a viewPort which is a sub-section of pixels within this RGB camera image. The annotated image is then converted into Bitmap format before being output to the device screen. This section of the paper explains the procedures performed to achieve this. The Java code explained in this section can be found in 1.1.1.1.Appendix C.

5.4.1 Permissions and features

The first step in accessing the video steam is specifying the camera permissions and application features within the Android manifest as is shown in Figure 5-2. This manifest extract shows that the application must gain permission to use the device camera from the Android operating system. It also specifies that the Android device used to run this application must have a rear facing camera which has autofocus capabilities. The rear facing camera is needed for this application as the user must be able to access the device screen while the camera views the upcoming crop rows.



```
<uses-permission android:name="android.permission.CAMERA" /> // Request permission for Camera access
<uses-feature android:name="android.hardware.camera" android:required="true" /> // a device with a camera is required
<uses-feature android:name="android.hardware.camera.autofocus" android:required="true" /> // camera autofocus is required
```

Figure 5-2: Android Manifest camera permissions

5.4.2 CameraPreview.java

This application accesses the camera hardware through the CameraPreview.java class shown in Appendix 1.1.1.1.C.3.

This class extends the ViewGroup.java class, which allows it to create interface layouts. The CameraPreview class also implements the SurfaceHolder.Callback interface, which is used to connect the camera hardware to the application. This class is responsible for passing image data to the main Row_Follow_Main.java class each time the camera captures a new frame. Because the Row_Follow_Main.java class implements Camera.PreviewCallback and has an onPreviewFrame() method, each time the camera gets a new frame the CameraPreview object will send the image data to this method. The onPreviewFrame() method then copies the incoming byte array, representing the new camera frame, to the processByte array used for processing the image.

5.4.3 Resolution, Framerate, and Camera class

The Row_Follow_Main.java class, found in Appendix 1.1.1.1.C.2, is where all other video stream related code is defined. The setUpApp() method within this class accesses the Android devices' rear facing camera by using the Camera.open() method.

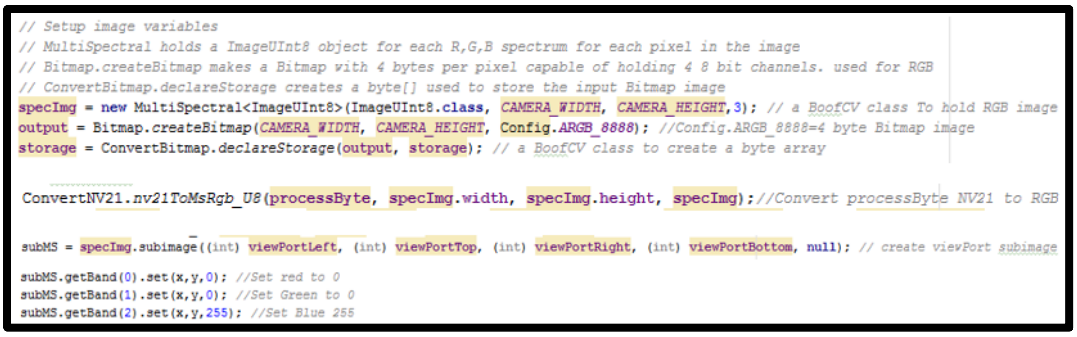
This method returns the rear facing camera by default. The camera resolution for the image returned is set to 320x240 at 30fps. This is achieved by accessing the Camera.Parameters.java class and is set using the global variables CAMERA_WIDTH and CAMERA_HEIGHT. 320x240 was selected as the resolution size as this offered a good balance between displaying a clear image and keeping resource usage to a minimum. 30fps was the frame rate selected because it was fast enough for data acquisition while not unnecessarily using the device's resources. This frame rate also shows a smooth transition between frames for the displayed image output. The Camera.java class was one of the classes that got deprecated at the end of 2014 with the release of Android SDK 21 and the new Camera2.java class. This application continues to use the deprecated Camera.java class over the newer Camera2.java class because of the larger amount of available resources regarding the Camera.java class and the fact that the device used for testing was running an older software version that only supported the Camera.java class.

5.4.4 Image format

The image is transferred into this application for processing by the device Camera through the CameraPreview.java class as described in section 5.4.2 . This image is in Android's NV21 YCrCb format. For ease of applying a threshold level to identify plant pixels the image is converted from the NV21 format into 8-bit unsigned RGB format using the nv21ToMsRgb_U8() method of the BoofCV ConvertNV21.java class. This RGB format displays each pixel in the image using three 8bit values representing the colours Red, Green, and Blue, with each having a colour depth range from 0 to 255. Each colour band is stored separately in a BoofCV ImageUInt8 object that is held within a MutliSpectral<ImageUInt8> object. Access to individual pixels is granted through the getBand() method of the MultiSpectral class.

After the image has been assessed using the row following algorithm it is converted into a Bitmap using the BoofCV ConvertBitmap.multiToBitmap() method and set to the background of the Canvas object that is used to output the image to the screen. This is done inside a synchronised code block because two separate threads have access to the output object.

Figure 5-3 below shows some code samples used within the application to access and convert between image formats. Further details can be seen in Appendix 1.1.1.1.C.2.

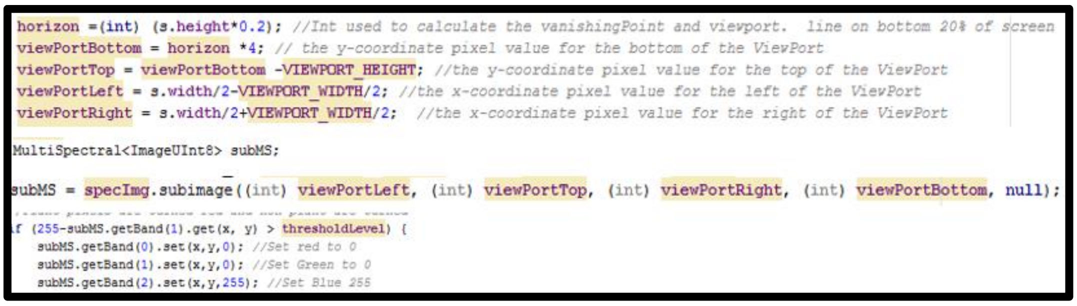


```
// Setup image variables
// MultiSpectral holds a ImageUInt8 object for each R,G,B spectrum for each pixel in the image
// Bitmap.createBitmap makes a Bitmap with 4 bytes per pixel capable of holding 4 8 bit channels. used for RGB
// ConvertBitmap.declareStorage creates a byte[] used to store the input Bitmap image
specImg = new MultiSpectral<ImageUInt8>(ImageUInt8.class, CAMERA_WIDTH, CAMERA_HEIGHT,3); // a BoofCV class To hold RGB image
output = Bitmap.createBitmap(CAMERA_WIDTH, CAMERA_HEIGHT, Config.ARGB_8888); //Config.ARGB_8888=4 byte Bitmap image
storage = ConvertBitmap.declareStorage(output, storage); // a BoofCV class to create a byte array

ConvertNV21.nv21ToMsRgb_U8(processByte, specImg.width, specImg.height, specImg);//Convert processByte NV21 to RGB

subMS = specImg.subimage((int) viewPortLeft, (int) viewPortTop, (int) viewPortRight, (int) viewPortBottom, null); // create viewPort subimage

subMS.getBand(0).set(x,y,0); //Set red to 0
subMS.getBand(1).set(x,y,0); //Set Green to 0
subMS.getBand(2).set(x,y,255); //Set Blue 255
```

Figure 5-3:Sample Android image format code

5.4.5 Viewport

A viewport was created using the sub-Image function from the BoofCV Image class. Creation of a sub-Image defines a portion of an image that can be independently used for calculations. This makes the sub-Image ideal for representing a viewport designed to straddle the edges of the crop rows as described in section 4.2 . The boundaries and position of this view port are defined in the setUpApp() method, and The viewport length and width are set by the global variables VIEWPORT_HEIGHT and VIEWPORT_WIDTH. In this application the viewport is 80x30 pixels, and is centred along the x-axis with its centre located at x coordinate 160 pixels which is half the width of the 320 pixel screen. The bottom of the viewport is located at y coordinate 192 which is 1/5 or 48 pixels from the bottom of the screen. Figure 5-4 below shows the program code used to create and access the viewport data and immediately following this code Figure 5-5 shows the resulting viewport displayed on the device screen.



```
horizon =(int) (s.height*0.2); //Int used to calculate the vanishingPoint and viewport.  line on bottom 20% of screen
viewPortBottom = horizon *4; // the y-coordinate pixel value for the bottom of the ViewPort
viewPortTop = viewPortBottom -VIEWPORT_HEIGHT; //the y-coordinate pixel value for the top of the ViewPort
viewPortLeft = s.width/2-VIEWPORT_WIDTH/2; //the x-coordinate pixel value for the left of the ViewPort
viewPortRight = s.width/2+VIEWPORT_WIDTH/2;  //the x-coordinate pixel value for the right of the ViewPort

MultiSpectral<ImageUInt8> subMS;

subMS = specImg.subimage((int) viewPortLeft, (int) viewPortTop, (int) viewPortRight, (int) viewPortBottom, null);

f (255-subMS.getBand(1).get(x, y) > thresholdLevel) {
   subMS.getBand(0).set(x,y,0); //Set red to 0
   subMS.getBand(1).set(x,y,0); //Set Green to 0
   subMS.getBand(2).set(x,y,255); //Set Blue 255
```

Figure 5-4: Creation and access of the viewport

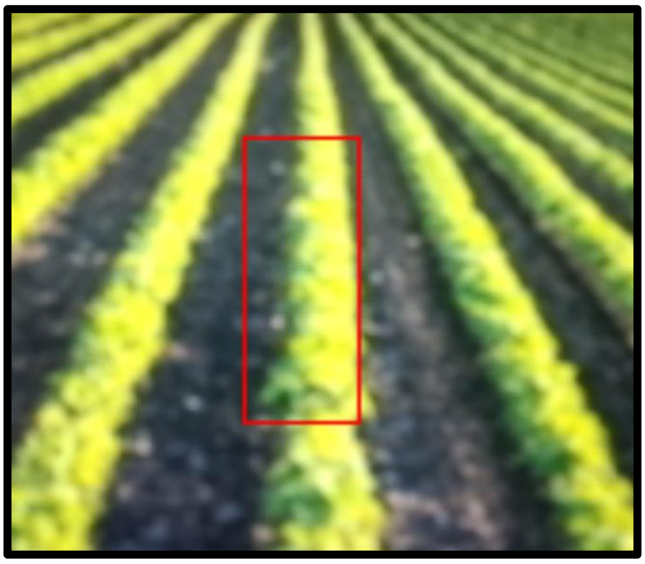


Figure 5-5: Viewport surrounding one crop row.

5.4.6 Plant Identification

Each pixel within the viewport window is compared against a settable threshold to check the greenness of each pixel. If the Green segment of the pixel is greater than or equal to the threshold, then the pixel is considered a plant and the pixel x and y coordinate values are then added to an ArrayList for further processing. In this application the original threshold level is set to 128. To help visualize which pixels are plant and which are not within the viewport window, all pixels identified as plant are changed red by setting the Green and Blue values to 0 and the Red value to 255, any pixel that is not a plant is turned blue by setting the Red and Green values to 0 and the Blue value to 255. A coded example of this can be seen in Figure 5-4 above, and in Figure 5-6 below a crop image with a viewport can be seen where the plant pixels are shown in red and the non-plant pixels in blue.

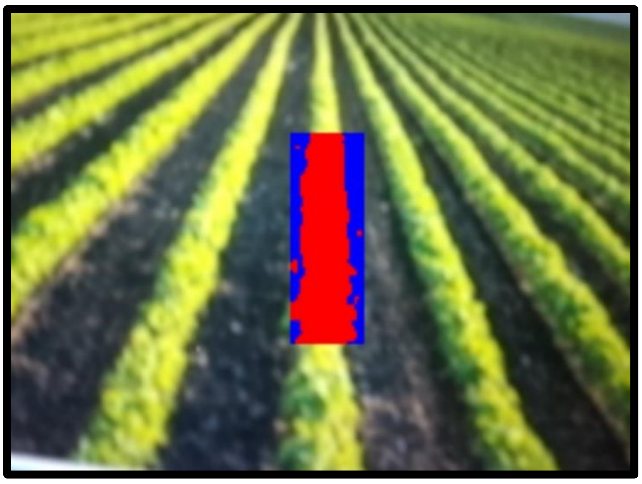


Figure 5-6: Viewport with plants identified in red

5.4.7 Threads

The application begins on one thread and another is created within the setUpApp() method to handle the time taken to process the video using the row finding algorithm. This is done because the calculations used to identify the crop rows within the image may take longer than the time taken for the cameraPreview to update with a new frame. Without a thread this causes a backlog of frames on this thread until the system runs out of memory and terminates. By creating seperate threads for processing and image acquisition each new frame that comes from the camera through the CameraPreview object is passed to the onPreviewFrame() method which updates the byteArray with the new frame data and sends a thread.interupt() message to let the processingThread class know that there is another frame available for processing. The separate thread then handles the new frame once it has finished with the previous frame. If a third frame arrives before the processorThread accesses the second waiting frame, the second frame is discarded and replaced with the third frame. The processorThread will miss processing frame two but this way no backlog of frames waiting to be processed will occur so the system will not terminate prematurely.

Because there are multiple threads running that have access to the same data, shared resources have to be accessed within a synchronised code block. This allows only one thread access to shared resources at any one time. If a synchronised code block is not used, then it could produce errors in the data.

5.5 Image annotation

The Android Canvas.java class was used to annotate the video image after processing had been done. The “output” Bitmap is set as the background of the Canvas which allows annotations to be drawn in the foreground. The image in this application has been annotated with a rectangle outlining the viewport, a line representing the regression line calculated, and some text. This was done using the drawRect(), drawLine(), and the drawText() methods of the Canvas class. The text displays the x_ alignment and slope_alignment correction data to be sent out to the Actuator control, the quality of fit value, and the vanishingPoint information. When the quality variable drops below 4, the regression line is no longer annotated to the screen. This simulates a live system reverting to manual control when the quality of fit is small. Once all annotated data is drawn onto the Bitmap image the mDraw.postInvalidate() method call tells the GUI to update the display to include the newly annotated Bitmap image.

5.6 Row identification algorithm

The following section covers how the NCEA row following algorithm was written using Android code. All of these algorithm calculations are performed on the processThread as described in section 5.4.7 Threads28.

5.6.1 Threshold

The threshold is used to identify the plant pixels according to the level of Green in the 8-bit RGB pixel as defined in section 5.4.6 . The program begins with this threshold level at 128 and it is raised or lowered to adjust for lighting fluctuations. To achieve this the operator sets the VIEWPORT_PLANT_PROPORTION global constant to identify the plants’ stage of development. As there is no user interface in this application this proportion value must be hard set in code before compilation. The plant pixels have already been identified and placed into an Arraylist called myxList. The ArrayList.size() method is then called to find number of plant pixels and the proportion of plant pixels within the viewport is then calculated and compared to the expected proportion listed in the global constant. The threshold is then lowered or raised accordingly. Figure 5-7 below shows that the threshold level has been adjusted from 128 of 70 for an expected plant density of 40% which is the default setting of the developed application.

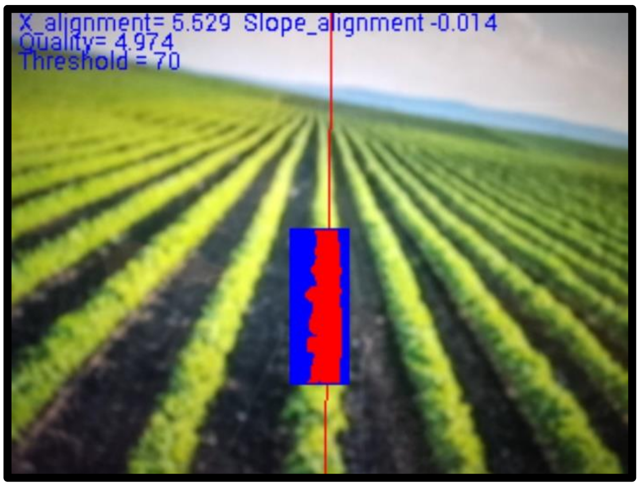


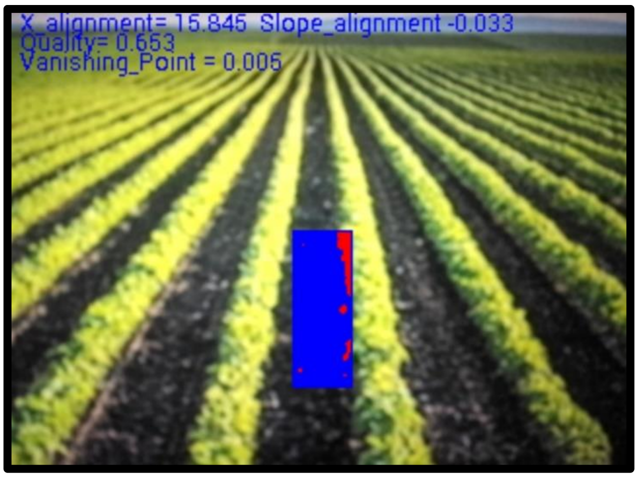
Figure 5-7:Threshold adjustment for an expected 40% plant density

5.6.2 Regression assessment

After the ArrayList holding all of the plant pixel coordinates is found, a call is made to the assess() method which in turn calls the fit() method used to calculate the regression line, correction data, and the new threshold value. If the fit() method deems the new data is of good quality then the assess() method will overwrite the old correction data held in global variables, with the new frames correction data as well as set up the data needed to draw the regression line.

5.6.3 Regression fit

The applications fit() method uses equations (4.3) and (4.4) to calculate the fitOffset and fitSlope values that will minimise the error in the cost function shown in equation (4.1). The fitOffset and fitSlope values then identify the regression line of best fit, as well as the correction values needed to achieve this minimum cost. If there are too few plant identified pixel values then these calculations will not be performed and subsequently the regression line and correction values will not be known or displayed. In addition to these calculations, this method also compares the plant pixels and there locations with the data from the last frame to calculate a quality of fit. The regression line is only drawn to the screen when the quality of fit is adequate. This value is set at 4 for this application. This application simulates the return to manual control for a poor quality fit by not annotating the regression line to the screen as shown in Figure 5-8.



```
X_alignment= 15.845 Slope_alignment -0.033
Quality= 0.653
Vanishing_Point = 0.005
```

Figure 5-8: A poor quality fit showing no regression line

5.6.4 Limits

The application has a limit() method that is used to apply limitation on the slope and vanishing point. These limits identify the boundaries of the regression line to ensure that the regression line and correction data is projected in the correct direction. This limit is set using a percentage value representing a percentage of pixel values for the boundary. This application has the limit set at 0.2.

5.7 Actuator control commands

Although physical actuator control was beyond the scope of this research the steering correction data that is to be sent to the actuator was calculated and displayed to the screen. This steering correction data was calculated using the methods found above in section 5.6 . As covered in section 4.4 the fitOffset and "vanishing point" are used to calculate the lateral movement steering corrections, while the corrections to the angular displacement are identified with the fitSlope variable. This information is annotated to the screen as X_ alignment, Vanishing_point, and Slope_alignment.

Chapter 6: Evaluation and optimisation

This section covers the testing methodology and optimisation techniques used for this research. As the agile programming method was used, each section of code produced in the research was written in small segments of code which were individually tested and optimised or discarded until the required result was found.

6.1 Test Equipment

The aim of this research was to see if an Android device was capable of implementing a computer vision algorithm such as the NCEA's row following algorithm so the written code was only tested using one specific test device. Further testing would need to be done using multiple devices to identify how well the application performed on differing hardware however that was beyond the scope of this research.

A Samsung Galaxy S4 was used for all module testing of this application. Samsung ('Samsung' 2015) listed the specifications for this android device as:

- Android OS 4.2.2
- Quad Core 1.9GHz processor
- Full HD Super AMOLED display
- 2GB RAM
- 16GB internal memory with up to 64GB external memory
- CMOS 13MP rear camera and CMOS 2MP front camera
- Sensors include Accelerometer, Geomagnetic, Gyroscope, Broadcom BCM47521 GLONASS GPS, AGPS, Bluetooth 4.0

Android code was uploaded to the device using the Android Studio and Intellij IDEs as explained in section 5.1 . The application utilised the test device's rear facing13MP camera to acquire the image before it was processed. The relevant test results were then displayed on the device's screen

6.2 Procedure

As stated in section 5.1 the agile programming method was used for writing and testing the software for this application. This allowed for testing of individual units which was essential during this project as the Android development kit and other Java libraries have a such vast number of libraries available and the development timeframe for this application didn't allow for an investigation into every aspect of code available. Useful and not useful code was identified quickly through this method. To identify the usefulness of a piece of code a small program was written or a sample program was investigated to identify what functions were available through the code use. During this initial discovery stage very little testing was done and more emphasis was set on finding code that may be useful in the future.

After some basic code was identified as being of possible use, further testing and investigation went into identifying uses for the code. Android developers list various automated testing tools to aid software development, however this project uses a manual testing method where unit tests for each piece of new code are manually performed and result are checked against predictable calculated values. Code was then edited and optimised or discarded were applicable.

6.3 Video stream access

After many hours researching how to access and process individual pixels from images captured by the Android device's camera, BoofCV was selected to access the video stream. This was due to its ease of use. BoofCV offered image conversion to a number of formats such as binary, greyscale, RGB, HSV, and BoofCV also offered a range of coded examples to be explored for different computer vision techniques. While OpenCV also contains a wide range of libraries for computer vision, BoofCV was selected as it was written entirely in Java which can be easily ported to an Android system.

For accessing the device camera the application used the Camera.open() method and not the BoofCV recommended cameraPreviewSetup() method. This was done for two reasons. The first being that only devices with rear facing cameras are supported as defined in the Android Manifest so Camera.open will always open the correct rear facing camera, and secondly to limit the memory used to run the application by limiting the amount of code used.

A 320x240 resolution image was had set for use and not BoofCV recommended closest() method to select resolution. This BoofCV method is more flexible as it offers a variety of different resolution option however 320x240 is a fairly standard resolution, and was one which was offered on the test device, so the hard coded option was selected to cut down on code used to write the application. Further application development for use on additional devices may need to include the BoofCV recommended closest() method.

Although different image formats were tested, RGB was selected for this application for its ease of implementation. The starting threshold for RGB was easily defined as G=128, and while other formats such as HSV are often more effective in varying lighting conditions, the use of the varying threshold and the density level within the viewport make this application very effective when dealing with light changes making RGB just as effective.

During plant pixel identification, only one loop is made through the entire viewport and useful plant pixels are stored in an ArrayList for further processing. This was changed from earlier unit tested models which included a loop to change pixel colour and a separate loop to calculate the regression line. Inclusion of the ArrayList allows processing of the plant identified pixels without the need for a second loop through the entire viewport. The ArrayList was selected to store the plant pixel coordinates over other List and Array options that are available in Java because the ArrayList size doesn't need to be defined prior to using the list which allows the one list to be used for any number of pixels. This is useful for this application as the actual number of plant pixels varies from frame to frame thus making a set size array impractical.

6.4 Image annotation

The built in Android Canvas class detailed in section 5.5 was used for this application as it is part of the Android SDK package and it offers a wide range of annotation options. The availability of multiple methods used for drawing rectangles for the view window, lines for the regression line, and written characters for information output was the main reason no other options for annotation were investigated.

6.5 Row identification algorithm

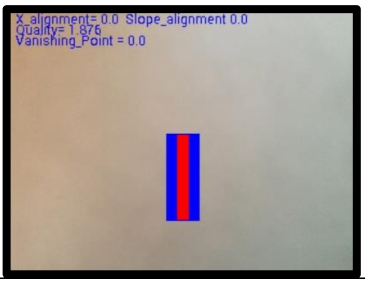
6.5.1 Accuracy

Testing for the accuracy of the row identification algorithm was performed by entering known predictable data into the algorithm and verifying the results shown on the screen. This testing involved inserting pixel co-ordinates that were deemed plant into the ArrayList() and verifying the output of these known pixel coordinates with the output of the regression line drawn and its associated correction data. Sample Accuracy test code can be found in 0 and results for this testing can be found below in Table 6-1.

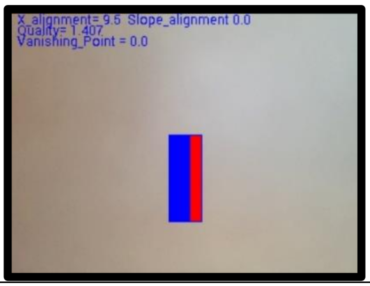
During this testing an error was found that identified an inaccuracy of half a pixel length when the plant pixels were situated over the positive x or right portion of the view window. The source of the code causing this error could not be located before this report was written so this slight error is listed as one of the limitation of this code. A possible cause of this error may be a miss calculate in the alignment of the viewport however this is just speculation and further investigation into the cause of this error is needed before the actual error can be eliminated.

Table 6-1: Accuracy tests for the row identification algorithm

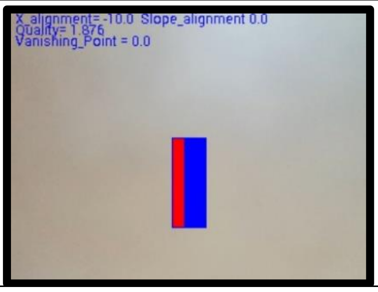
Note: For this accuracy test the quality value was disabled in the code for all scenarios except scenario 5 which was testing the quality value.		
Scenario 1: A straight crop row indicating the tractor is on the correct path		
Notes: This shows zero for all fields as expected.		
Predicted Output	**Actual Output**	**Test Image**

X_alignment: 0.0
Slope_alignment: 0.0
Vanishing_point: 0.0
	X_alignment: 0.0
	Slope_alignment: 0.0
	Vanishing_point: 0.0
		
		X_alignment= 0.0 Slope_alignment 0.0
		Quality= 1.876
		Vanishing_Point = 0.0

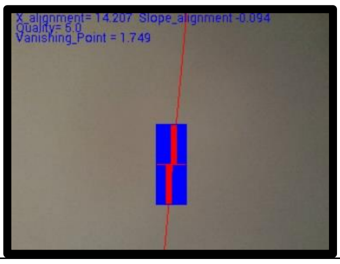
Scenario 2: The row is positioned to the right side of the viewport making the tractor too far to the left of the row		
Notes: The alignment of the tractor is straight but offset negatively making the lateral alignment negative with the slope zero. There was an error noted in this test that showed the x_alignment at 9.5 when the calculated value was 10. The cause of this error is yet to be determined.		
Predicted Output	**Actual Output**	**Test Image**

X_alignment: 10.0
Slope_alignment: 0.0
Vanishing_point: 0.0
	X_alignment: 9.5
	Slope_alignment: 0.0
	Vanishing_point: 0.0
		
		X_alignment= 9.5 Slope_alignment 0.0
		Quality= 1.407
		Vanishing_Point = 0.0

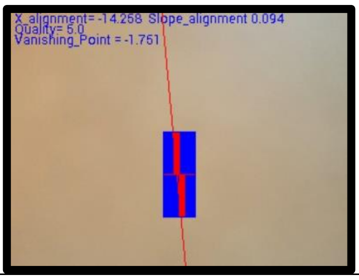
Scenario 3: The row is positioned to the left side of the viewport making the tractor too far to the right of the row		
Notes: This scenario is similar to scenario 2 however the tractor is now displaced in the positive range of the x-axis. This would make the predicted correction data the same as scenario 2 but negated. The predicted output is the same as the actual output.		
Predicted Output	**Actual Output**	**Test Image**

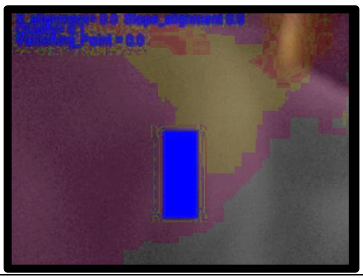
X_alignment: -10.0
Slope_alignment: 0.0
Vanishing_point: -0.0
	X_alignment: -10.0
	Slope_alignment: 0.0
	Vanishing_point: -0.0
		
		X_alignment= -10.0 Slope_alignment 0.0
		Quality= 1.876
		Vanishing_Point = 0.0

Scenario 4: The row is in the bottom left and top right of the viewport simulating the tractor turned in the anti-clockwise direction.		
Notes: This scenario is aimed to test corrections to the aggregate slope of the tractor. The tractor tilted in the clockwise direction would generate a negative value. With a tilt as shown in the image, a slope correction of a -0.09 can be expected.		
Predicted Output	**Actual Output**	**Test Image**

Slope_alignment: -0.09	Slope_alignment: -0.094	 X_alignment= 14.207 Slope_alignment -0.094
		Quality= 5.0
		Vanishing_Point = 1.749

Scenario 5: The row is in the bottom right and top left of the viewport simulating the tractor turned in the clockwise direction		
Notes: This scenario is aimed to test corrections to the aggregate slope of the tractor. The tractor tilted in the clockwise direction would generate a positive value. With a tilt as shown in the image, a slope correction of a 0.09 can be expected.		
Predicted Output	**Actual Output**	**Test Image**

Slope_alignment: 0.09	Slope_alignment: 0.094	 X_alignment= -14.258 Slope_alignment 0.094
		Quality= 5.0
		Vanishing_Point = -1.751

Scenario 6: The front of the tractor is off course with a poor quality fit.		
Notes: This scenario is aimed to test the quality value. With the quality value below 4 the image will disable the regression line simulating the system reverting to manual control.		
Predicted Output	**Actual Output**	**Test Image**
Quality: <4 No regression line output	Quality: No regression line output	

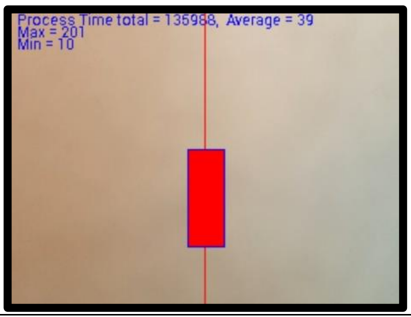
6.5.2 Processing

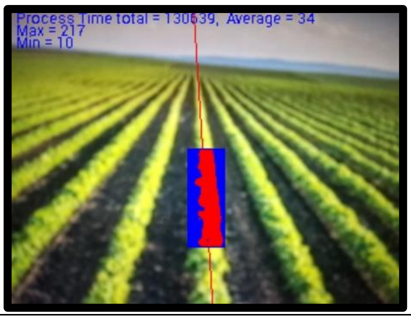
The processor speed was calculated using the Java System method System.currentTimeMilis(). The system time was checked at the start and end of the process thread and the difference was calculated to get the system time. This data was stored to obtain the average, maximum, and minimum process times.

Firstly the view window was populated with all pixels within the viewport identified as plant as this would take the longest time for the system to process data. The system was then left to run for a two minutes while data was collected. A second test was then run to test system processing speed using actual data that would be obtained during a typical free system run. Table 6-2 below shows results from these two tests. As expected the worst case scenario performed slower calculations with an average of 39 ms per frame, while the free running test got a better result at 34 ms per frame. Both tests resulted in larger than expected Maximum processing time and lower than expected minimum processing times. A hypothesis for the maximum values was the slow speed of the system during start up, however when this test was re-performed with a 5 and 10 second lag before calculating data it made no difference in the minimum and maximum results.

Table 6-2: Processor speed tests

Processor Test	**Results (ms)**	**Test Image**

Worst case scenario	Average: 39
	Maximum: 201
	Minimum: 10
		 Process Time total = 135988, Average = 39
		Max = 201
		Min = 10

Free running system	Average: 34
	Maximum: 217
	Minimum: 10
		 Process Time total = 130639, Average = 34
		Max = 217
		Min = 10

Another hypothesis for the range in processing times is the application at times using larger than normal amounts of system RAM which would slow down the processing speed leading to the high maximum processing time. When verifying the systems RAM usage fluctuations in the amount of RAM used as can be seen as shown in Figure 6-1. This is one possible cause in the high maximum processing time however the actual cause of this high value was not determined at the time of writing this paper.

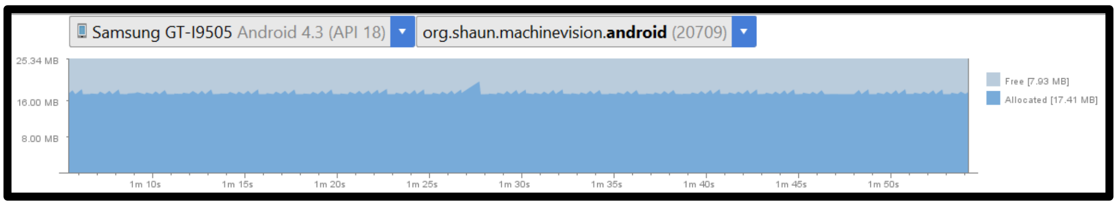


Figure 6-1: RAM usage

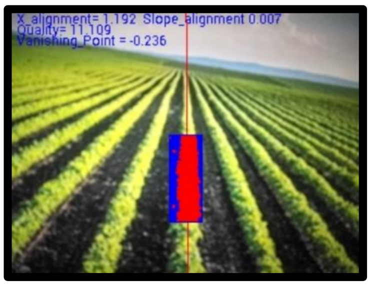
Further minimisation for the processor time could be achieved by processing less data, which could be done by either creating a smaller viewport window or by keeping the viewport window the same size but only processing every second or third row of pixels. This second option would be the optimum method as it would give access to a wider range of data than concentrating the data in a smaller viewport. Testing for these two methods of operation had not been performed at the time of writing this paper.

6.6 Actuator control commands

After testing the data for accuracy and speed using know pixel inputs the application was tested using live data obtained from the test devices camera. Each of the scenarios in Table 6-3 below show values calculated for the actuator control outputs using random live data from the Android camera. In each case the outputs were predicted and compared with the actual output given by the application. and the resulting figures that were displayed to the screen were visually checked to see if they matched the predicted output. As the data used in this test was random outputs had to be visually predicted rather than calculated. While the vanishing point figure appears on the screen testing for its accuracy could not be performed in time for inclusion in this document as identified in section 5.3.2 .

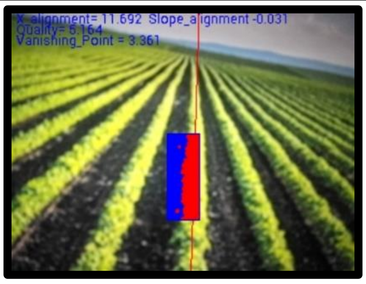
Table 6-3:Actuator control simulations

Scenario 1: The tractor is heading in a nearly straight line on the correct path aligned with a crop row.		
Notes: In this scenario no corrections need to be made so all fields should be close to zero except the quality field which should be good at around 10.		
Predicted Output	**Actual Output**	**Test Image**

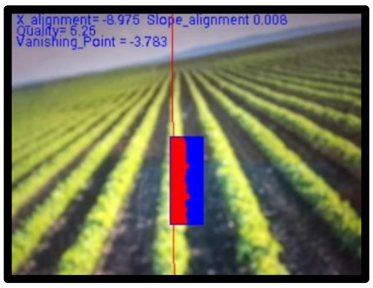
Quality: 10.0
X_alignment: 0.0
Slope_alignment: 0.0
Vanishing_point: 0.0
	Quality: 11.10
	X_alignment: 1.19
	Slope_alignment: 0.007
	Vanishing_point: -0.24
		 X_alignment= 1.192 Slope_alignment 0.007
		Quality= 11.109
		Vanishing_Point = -0.236

Scenario 2: The tractor is heading in a straight line but displaced to the left of the crop row.		
Notes: In this scenario the tractor is heading in a straight line so the angular displacement should be close to zero which would make the Slope_allignment correction data close to zero. The tractor is aligned too far to the left of the row putting it in the negative range of the x-axis with half of the viewport on the row and half off of the row. As the viewport is 30 pixels wide a lateral correction of about 15 pixels in the positive direction needs to be made. The Quality value is only about half of what it should be which would make the predicted value close to 5.		

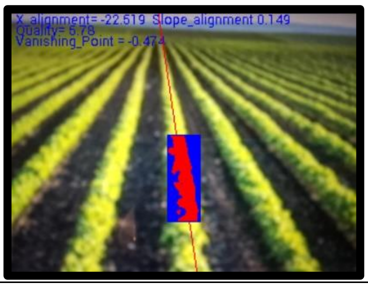
Predicted Output	**Actual Output**	**Test Image**

Quality: 5.0
X_alignment: 15.0
Slope_alignment: 0.0
Vanishing_point: 3.0
	Quality: 5.14
	X_alignment: 11.69
	Slope_alignment: -0.031
	Vanishing_point: 3.36
		 X_alignment= 11.692 Slope_alignment -0.031
		Quality= 5.164
		Vanishing_Point = 3.361

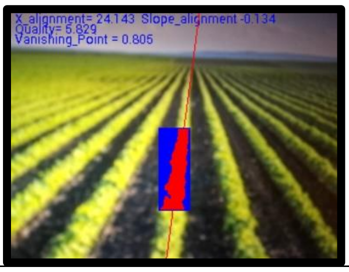
Scenario 3: The tractor is heading in a straight line but displaced to the right of the crop row.		
Notes: This scenario is similar to scenario 2 however the tractor is now displaced in the positive range of the x-axis. This would make the predicted correction data the same as scenario 2 but negated, with a quality rating of about 5.		
Predicted Output	**Actual Output**	**Test Image**

Quality: 5.0
X_alignment: -15.0
Slope_alignment: 0.0
Vanishing_point: -3.0
	Quality: 5.25
	X_alignment: -8.98
	Slope_alignment: 0.008
	Vanishing_point: -3.78
		 X_alignment= -8.975 Slope_alignment 0.008
		Quality= 5.25
		Vanishing_Point = -3.783

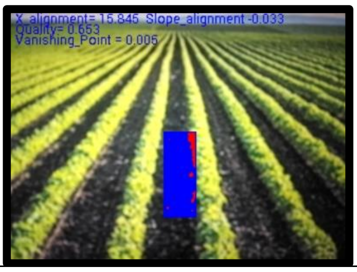
Scenario 4: The front of the tractor is tilted in a clockwise direction.		
Notes: This scenario is aimed to test corrections to the aggregate slope of the tractor. The tractor tilted in the clockwise direction would generate a positive value. With a tilt as shown in the image, a slope correction of a 0.1 can be expected.		
Predicted Output	**Actual Output**	**Test Image**

Quality: 5.0
Slope_alignment: 0.1
	Quality: 5.78
	Slope_alignment: 0.149
		 X_alignment= -22.519 Slope_alignment 0.149
		Quality= 5.78
		Vanishing_Point = -0.474

Scenario 5: The front of the tractor is tilted in an anti-clockwise direction.		
Notes: This scenario is aimed to test corrections to the aggregate slope of the tractor in the opposite direction as scenario 4. This should result in similar corrections in the slope in the negative direction.		
Predicted Output	**Actual Output**	**Test Image**

Quality: 5.0
Slope_alignment: -0.1
	Quality: 5.83
	Slope_alignment: -0.134
		 X_alignment= 24.143 Slope_alignment -0.134
		Quality= 5.829
		Vanishing_Point = 0.805

Scenario 6: The front of the tractor is off course with a poor quality fit.		
Notes: This scenario is aimed to test the quality value. With the quality value dropping below 4 the image will disable the regression line simulating the system reverting to manual control.		
Predicted Output	**Actual Output**	**Test Image**

Quality: <4
No regression line output
	Quality: 0.653
	No regression line output
		 X_alignment= 15.845 Slope_alignment -0.033
		Quality= 0.653
		Vanishing_Point = 0.005

6.7 Code optimisation

To minimise memory usage in an attempt to speed up processing speed during the full system test the Android code was optimised using Intellij's built in code cleanup tool. This helped identify any unused code and support libraries that could be deleted and gave suggestions on how to optimise the performance of the code. After the code was cleaned up the system RAM usage was identified by looking at the system logcat files. Outputs for the RAM memory monitor and the Dalvik garbage collection data both show a system RAM usage of around 17MB. These outputs are shown in Figure 6-2 and Figure 6.3.

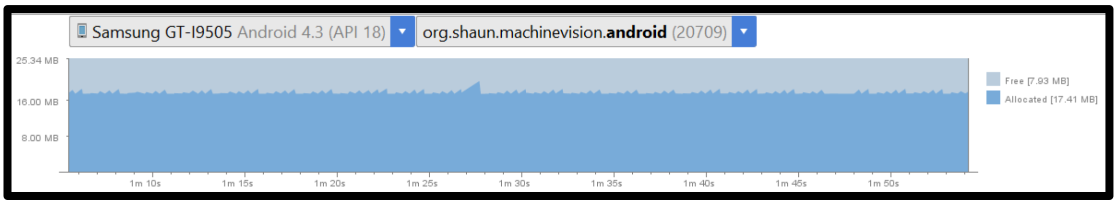
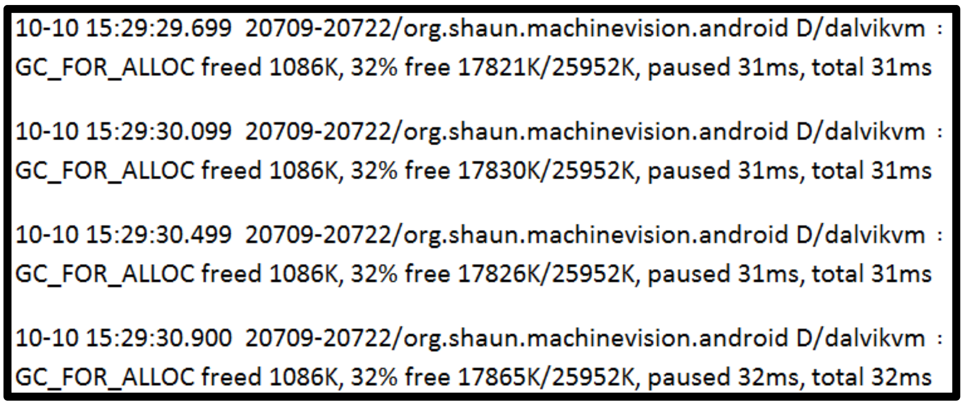


Figure 6-2: RAM usage after code optimisation



```
10-10 15:29:29.699  20709-20722/org.shaun.machinevision.android D/dalvikvm :
GC_FOR_ALLOC freed 1086K, 32% free 17821K/25952K, paused 31ms, total 31ms

10-10 15:29:30.099  20709-20722/org.shaun.machinevision.android D/dalvikvm :
GC_FOR_ALLOC freed 1086K, 32% free 17830K/25952K, paused 31ms, total 31ms

10-10 15:29:30.499  20709-20722/org.shaun.machinevision.android D/dalvikvm :
GC_FOR_ALLOC freed 1086K, 32% free 17826K/25952K, paused 31ms, total 31ms

10-10 15:29:30.900  20709-20722/org.shaun.machinevision.android D/dalvikvm :
GC_FOR_ALLOC freed 1086K, 32% free 17865K/25952K, paused 32ms, total 32ms
```

Figure 6-3: Dalvik garbage collector output after code optimisation

Chapter 7: Results

This section of the dissertation records the key outcomes of this paper and how they match up with the desired project objectives.

7.1 Objectives and outcomes

This project has resulted in the development of a demonstration application that has capable of keeping RAM usage to an average of about 17 MB while processing the 30 fps, 320x240 resolution video, in an average of 34 ms per frame during typical running circumstances. Data showing the identifying a row of crows and steering correction data is also printed to the device screen.

This project has identified the capabilities of an Android device being used for farm vehicle automation by testing its ability to process a row finding application with speed and accuracy. The project has shown that an Android device can identify crop rows based on the greenness of the image pixel. Further development of Android based machine vision could dramatically reduce the cost and complexity of setting up alternative automated vehicle guidance systems**.**

7.1.1 SDK investigation

The initial objective of this project was the investigation into the Android SDK and various computer vision libraries which resulted in the use of BoofCV with the Intellij IDE for this project. BoofCV was used primarily for image conversion between different formats while the Android SDK was used elsewhere throughout the code.

7.1.2 Video stream access

Video stream access and manipulation were the second and third objectives for this project and they were made possible through the use of the BoofCV libraries. The android camera returned each frame to the main application in the NV21 format using a callback function within the CameraPreview class. This NV21 frame was then converted into a Multispectrum format containing 3 unsigned 8 bit integer image objects for each of the RGB colours. A portion of the RGB frame was then assigned a viewport and each pixel was within that viewport was compared against a threshold to identify it as plant or not and the colour of the pixel was changed accordingly. Plant pixels were stored into an ArrayList and sent for further processing. After processing the RGB image was converted to Bitmap format where it was annotated and sent to the device screen for display. This was achieved at 30fps with a resolution of 320x240.

The threshold level was also automatically changed during this process to allow for fluctuations in image lighting.

7.1.3 Image annotation

Image annotation was the fourth objective of this project and it was achieved using the Android Canvas class. Posting the main camera Bitmap image on a Canvas allowed for the vast array of methods within the Canvas class to be used to annotate the main image. Annotation used the drawText() method to write characters to the screen, the drawLine() method to draw the regression line on the screen, and the drawRect() method to identify the location of the viewport.

7.1.4 Row identification algorithm

The fifth objective of row identification used the ArrayList of plant pixels and the assess() and fit() methods to calculate the regression line and correction data, check for a quality of fit, and assign correction data and application output using these parameters. The fit() method used the NCEA row identification algorithm to obtain the regression and correction data. The quality of fit calculation compared the results of the previous frame to the current one and updated results accordingly.

Testing of for speed and accuracy of this data resulted in finding an inaccuracy of half a pixel length and with an average process time of 34 ms per frame during typical circumstances. Further work in this area needs to be done to find the cause of the data inaccuracy and the addition of extra viewports to check the accuracy of the viewpoint calculations.

7.1.5 Actuator control commands

The display for the actuator control simulation for the correction data was the final objective of this project. This was partially achieved and tested using number data output to the device screen. Initial testing with output numbers identified that these steering correction calculations were correct however further work in this area still needs to be done. A simulated steering device, such as the suggested sliding bar on the bottom of the display, would be a good way to test the steering correction data.

Chapter 8: Conclusion

This dissertation covered the development of an Android based vision guidance system for a tractor. The original specification for a system to include vision guidance data as well as other Android sensors, such as GPS, was revised after the project preliminary report. The new revision specified a vision based guidance system without any additional sensors. While this paper only involved the early stages of development of such an application, further development could lead to accessible tractor automation for everyone due to the ease of use and installation, as well as the low cost price of such an Android operated system. Listed below are the key findings of this project and further work that is needed to make a workable Android based tractor automation system.

8.1 Key project findings

This project developed an Android system that is capable of identifying rows of crops. While all of the project testing and simulation carried out during this project was based in a laboratory the work done has shown the potential use of Android devices in vehicle automation in the future.

The demonstration application that has been developed has kept RAM usage to an average of about 17 MB while processing the 30 fps, 320x240 resolution image, in an average of 34 ms per frame during typical circumstances.

This demonstrator application displayed a regression line to the screen to visually identify the crop row as well as outputting numerical values representing steering corrections to be sent to the actuator. The application also included a threshold level to deal with fluctuations in changing lighting conditions.

Due to time constraints the demonstrator application only uses one viewport window for calculations and does not include any additional viewports as described in section 4.3 . The use of only one viewport has meant that testing of the vanishing point for steering corrections could not be achieved. As defined during testing, there is also a problem with the code which outputs steering corrections that are incorrect by half a pixel. At the time of publication this error had not been resolved.

8.2 Further Work

Future work for this project would involve the inclusion of additional viewport windows in the Android software, which would be followed up with the system outputting a signal to an actuator that controls the vehicle. All of these devices would then need to be field tested.

List of References

Abeles, P 2012, 'BoofCV', viewed 6 July 2015, <http://boofcv.org/>.

Abidine, A, Heidman, B, Upadhyaya, S & Hills, D 2002, 'Application of RTK GPS based auto-guidance system in agricultural production', *ASAE Paper Nº 021152–ASAE. St Joseph MI*.

'Android Developers', 2015, viewed 4 April 2015, <http://developer.android.com/index.html>.

Australian Bureau of Statistics 2013, 'Land Management and Farming in Australia 2012-2013', *ABS Catalogue No. 4627.0, Commonwealth of Australia*.

Bell, T 2000, 'Automatic tractor guidance using carrier-phase differential GPS', *Computers and Electronics in Agriculture*, vol. 25, no. 1, pp. 53-66.

Benson, E, Stombaugh, T, Noguchi, N, Will, J & Reid, J 1998, 'An evaluation of a geomagnetic direction sensor for vehicle guidance in precision agriculture applications', *ASAE paper*, vol. 983203.

Billingsley, J & Schoenfisch, M 1997, 'The successful development of a vision guidance system for agriculture', *Computers and Electronics in Agriculture*, vol. 16, no. 2, pp. 147-63.

Borenstein, J 1998, 'Experimental results from internal odometry error correction with the OmniMate mobile robot', *IEEE Transactions on Robotics & Automation*, vol. 14, no. 6, pp. 963-9, <http://ezproxy.usq.edu.au/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=a9h&AN=93052690&site=ehost-live>.

Brooke, D 1972, 'Operating Experience with Wide-wire Leader Cable Tractors, ASAE Paper 72-119', *American Society of Agricultural Engineers, St. Joseph, MI*.

Defense", UDO 2008, 'Global Positioning System Standard Positioning Service Performance Standard'.

Dingsoyr, T 2010, 'Agile Software Development', viewed 19/7/2015.

Emmi, L, Gonzalez-de-Soto, M, Pajares, G & Gonzalez-de-Santos, P 2014, 'Integrating Sensory/Actuation Systems in Agricultural Vehicles', *Sensors (14248220)*, vol. 14, no. 3, pp. 4014-49, <http://ezproxy.usq.edu.au/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=a9h&AN=95106702&site=ehost-live>.

FAO 2009, 'FAO's Director-General on How to Feed the World in 2050', *Population and Development Review*, vol. 35, no. 4, pp. 837-9, <http://www.jstor.org/stable/25593700>.

Gan-Mor, S & Clark, R 2001, 'DGPS-based automatic guidance-implementation and economical analysis', in *ASAE Meeting Paper*: *proceedings of theASAE Meeting Paper*.

Gan-Mor, S, Clark, RL & Upchurch, BL 2007, 'Implement lateral position accuracy under RTK-GPS tractor guidance', *Computers and Electronics in Agriculture*, vol. 59, no. 1, pp. 31-8.

Gerrish, JB, Stockman, G, Mann, L & Hu, G 1985, 'Image processing for path-finding in agricultural field operations', *Paper-American Society of Agricultural Engineers (USA). Microfiche collection. no. fiche no. 85-3037.*

'GPS.Gov', 2015, viewed 11 April 2015, <http://www.gps.gov/>.

Han, S, Zhang, Q, Ni, B & Reid, JF 2004, 'A guidance directrix approach to vision-based vehicle guidance systems', *Computers and Electronics in Agriculture*, vol. 43, no. 3, pp. 179-95, <http://www.sciencedirect.com/science/article/pii/S0168169904000286>.

'IEEE Standard Glossary of Software Engineering Terminology', 1990, *IEEE Std 610.12-1990*, pp. 1-84.

Jiang, G, Wang, Z & Liu, H 2015, 'Automatic detection of crop rows based on multi-ROIs', *Expert Systems with Applications*, vol. 42, no. 5, pp. 2429-41, <http://ezproxy.usq.edu.au/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=a9h&AN=100062818&site=ehost-live>.

Keskın, M, Say, SM & Görücü Keskin, S 2009, 'Evaluation of a Low-Cost GPS Receiver for Precision Agriculture Use in Adana Province of Turkey', *Düşük Maliyetli Bir GPS Alıcısının Adana İlinde Hassas Uygulamal&0131; Tarımda Kullanılabilirliğinin Değerlendirilmesi.*, vol. 33, no. 1, pp. 79-88, <http://ezproxy.usq.edu.au/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=a9h&AN=37196273&site=ehost-live>.

Kise, M, Noguchi, N, Ishii, K & Terao, H 2002, 'The development of the autonomous tractor with steering controller applied by optimal control', *Proceeding of the Automation Technology for Off-Road Equipment. Chicago-IL*.

Larsen, W, Nielsen, G & Tyler, D 1994, 'Precision navigation with GPS', *Computers and Electronics in Agriculture*, vol. 11, no. 1, pp. 85-95.

Martin, RC 2003, *Agile Software Development: Principles, Patterns, and Practices*, Prentice Hall PTR.

Ming, L, Imou, K, Wakabayashi, K & Yokoyama, S 2009, 'Review of research on agricultural vehicle autonomous guidance', *International Journal of Agricultural & Biological Engineering*, vol. 2, no. 3, pp. 1-16, <http://ezproxy.usq.edu.au/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=a9h&AN=47114795&site=ehost-live>.

Moore, T, Tilak, S, Papadouplous, P & Clementi, L 2014, 'Programmatically defining the software footprint of sensor networks using the Android platform', *Software: Practice and Experience*, vol. 44, no. 10, pp. 1277-86, <http://dx.doi.org/10.1002/spe.2209>.

Morgan, K 1958, 'A step towards an automatic tractor', *Farm mech*, vol. 10, no. 13, pp. 440-1.

Noguchi, N & Terao, H 1997, 'Path planning of an agricultural mobile robot by neural network and genetic algorithm', *Computers and Electronics in Agriculture*, vol. 18, no. 2, pp. 187-204.

Noguchi, N, Kise, M, Ishii, K & Terao, H 2002, 'Field automation using robot tractor', in *Proceedings of Automation Technology for off-road equipment*: *proceedings of theProceedings of Automation Technology for off-road equipment* pp. 239-45.

OpenCV 2015, *OpenCV*, viewed 6 May 2015, <http://opencv.org/>.

Palmer, R & Matheson, S 1988, 'Impact of navigation on farming', *American Society of Agricultural Engineers (Microfiche collection)(USA)*.

Pérez-Ruiz, M, Slaughter, DC, Gliever, CJ & Upadhyaya, SK 2012, 'Automatic GPS-based intra-row weed knife control system for transplanted row crops', *Computers & Electronics in Agriculture*, vol. 80, pp. 41-9, <http://ezproxy.usq.edu.au/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=a9h&AN=70364317&site=ehost-live>.

Pérez-Ruíz, M, Slaughter, DC, Fathallah, FA, Gliever, CJ & Miller, BJ 2014, 'Co-robotic intra-row weed control system', *Biosystems Engineering*, vol. 126, pp. 45-55, <http://ezproxy.usq.edu.au/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=a9h&AN=97846736&site=ehost-live>.

Reid, J, Searcy, S & Babowicz, R 1985, 'Determining a guidance directix in row crop images', *American Society of Agricultural Engineers (Microfiche collection)(USA)*.

Reid, JF, Zhang, Q, Noguchi, N & Dickson, M 2000, 'Agricultural automatic guidance research in North America', *Computers and Electronics in Agriculture*, vol. 25, no. 1, pp. 155-67.

Rierson, L 2013, *Developing Safety-Critical Software: A Practical Guide for Aviation Software*, CRC Press, America.

Rovira-Más, F, Zhang, Q, Reid, JF & Will, JD 2003, 'Machine Vision Based Automated Tractor Guidance', *International Journal of Smart Engineering System Design*, vol. 5, no. 4, pp. 467-80, viewed 2015/05/06, <http://dx.doi.org/10.1080/10255810390445300>.

'Samsung', 2015, viewed 4 April 2015, <http://www.samsung.com/>.

Searcy, SW, Schueller, JK, Bae, YH & Stout, BA 1990, 'Measurement of agricultural field location using microwave frequency triangulation', *Computers and Electronics in Agriculture*, vol. 4, no. 3, pp. 209-23.

Shen, WEI & Liu, G 2007, 'A robust approach to obtain a guidance directrix for a vision-based agricultural vehicle guidance system', *New Zealand Journal of Agricultural Research*, vol. 50, no. 5, pp. 1067-72, <http://ezproxy.usq.edu.au/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=a9h&AN=35522088&site=ehost-live>.

Subramanian, V, Burks, TF & Arroyo, AA 2006, 'Development of machine vision and laser radar based autonomous vehicle guidance systems for citrus grove navigation', *Computers & Electronics in Agriculture*, vol. 53, no. 2, pp. 130-43, <http://ezproxy.usq.edu.au/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=a9h&AN=22082121&site=ehost-live>.

Sun, H, Slaughter, D, Ruiz, MP, Gliever, C, Upadhyaya, S & Smith, R 2010, 'RTK GPS mapping of transplanted row crops', *Computers and Electronics in Agriculture*, vol. 71, no. 1, pp. 32-7.

Tillett, N 1991, 'Automatic guidance sensors for agricultural field machines: a review', *Journal of agricultural engineering research*, vol. 50, pp. 167-87.

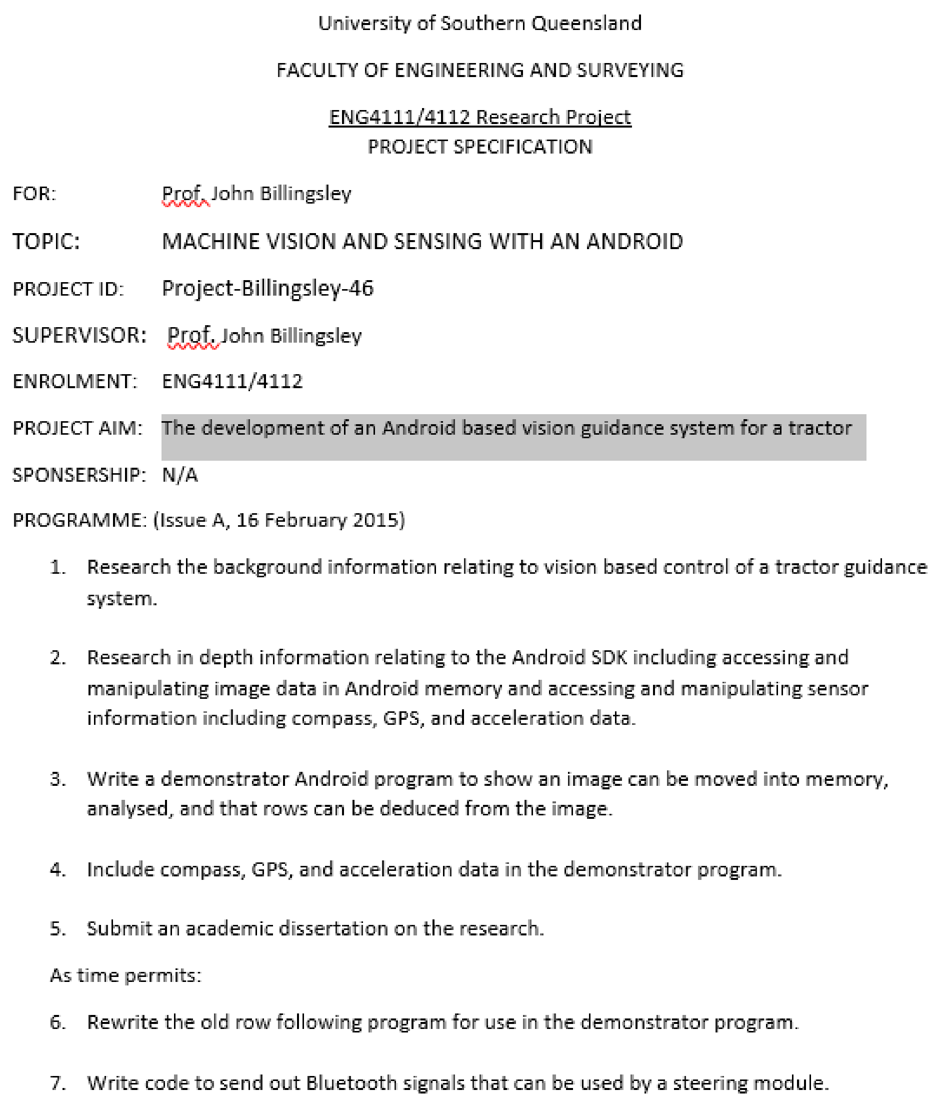
Wilson, JN 2000, 'Guidance of agricultural vehicles — a historical perspective', *Computers and Electronics in Agriculture*, vol. 25, no. 1–2, pp. 3-9, <http://www.sciencedirect.com/science/article/pii/S0168169999000526>.

Zhang, H, Cheng, B & Zhang, L 2008, 'DETECTION ALGORITHM FOR CROP MULTI-CENTERLINES BASED ON MACHINE VISION', *Transactions of the ASABE*, vol. 51, no. 3, pp. 1089-97, <http://ezproxy.usq.edu.au/login?url=http://search.ebscohost.com/login.aspx?direct=true&db=a9h&AN=34095197&site=ehost-live>.

Appendix A Project Specification

A.1. Original project specification

The original project specification is defined below. It identifies the original project specifications which were altered after the project preliminary report was marked.



University of Southern Queensland

FACULTY OF ENGINEERING AND SURVEYING

ENG4111/4112 Research Project
PROJECT SPECIFICATION

FOR:	Prof. John Billingsley
TOPIC:	MACHINE VISION AND SENSING WITH AN ANDROID
PROJECT ID:	Project-Billingsley-46
SUPERVISOR:	Prof. John Billingsley
ENROLMENT:	ENG4111/4112
PROJECT AIM:	The development of an Android based vision guidance system for a tractor
SPONSERSHIP:	N/A

PROGRAMME: (Issue A, 16 February 2015)

1. Research the background information relating to vision based control of a tractor guidance system.

2. Research in depth information relating to the Android SDK including accessing and manipulating image data in Android memory and accessing and manipulating sensor information including compass, GPS, and acceleration data.

3. Write a demonstrator Android program to show an image can be moved into memory, analysed, and that rows can be deduced from the image.

4. Include compass, GPS, and acceleration data in the demonstrator program.

5. Submit an academic dissertation on the research.

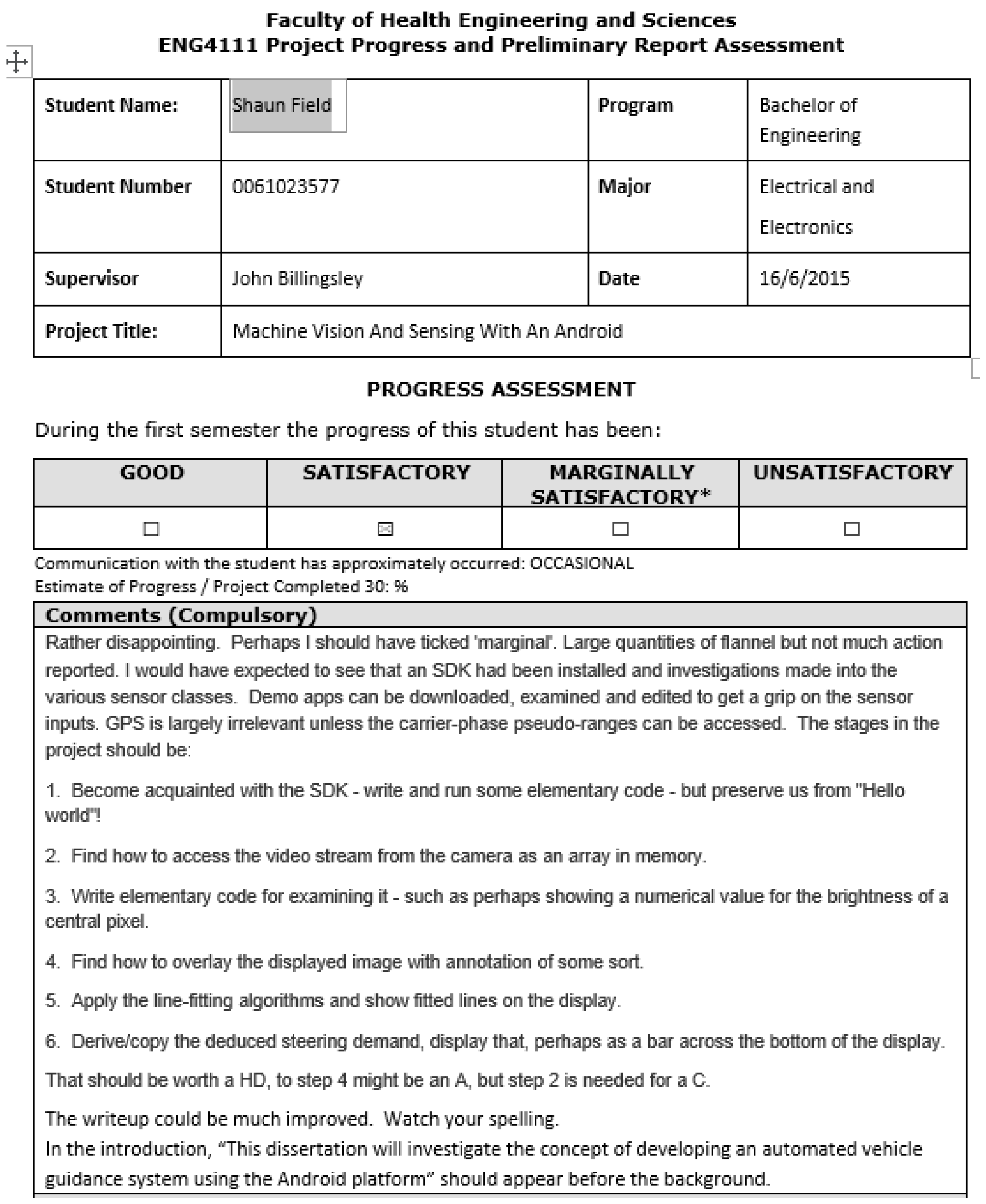
As time permits:

6. Rewrite the old row following program for use in the demonstrator program.

7. Write code to send out Bluetooth signals that can be used by a steering module.

A.2. Preliminary report recommended project stages

The project specification was altered after the preliminary report was marked. The new project specification of recommended stages for project completion was then followed to complete the project.



Faculty of Health Engineering and Sciences
ENG4111 Project Progress and Preliminary Report Assessment

Student Name:	Shaun Field	**Program**	Bachelor of Engineering
Student Number	0061023577	**Major**	Electrical and Electronics
Supervisor	John Billingsley	**Date**	16/6/2015
Project Title:	Machine Vision And Sensing With An Android		

PROGRESS ASSESSMENT

During the first semester the progress of this student has been:

GOOD	SATISFACTORY	MARGINALLY SATISFACTORY*	UNSATISFACTORY
☐	☒	☐	☐

Communication with the student has approximately occurred: OCCASIONAL
Estimate of Progress / Project Completed 30: %

Comments (Compulsory)

Rather disappointing. Perhaps I should have ticked 'marginal'. Large quantities of flannel but not much action reported. I would have expected to see that an SDK had been installed and investigations made into the various sensor classes. Demo apps can be downloaded, examined and edited to get a grip on the sensor inputs. GPS is largely irrelevant unless the carrier-phase pseudo-ranges can be accessed. The stages in the project should be:

1. Become acquainted with the SDK - write and run some elementary code - but preserve us from "Hello world"!

2. Find how to access the video stream from the camera as an array in memory.

3. Write elementary code for examining it - such as perhaps showing a numerical value for the brightness of a central pixel.

4. Find how to overlay the displayed image with annotation of some sort.

5. Apply the line-fitting algorithms and show fitted lines on the display.

6. Derive/copy the deduced steering demand, display that, perhaps as a bar across the bottom of the display.

That should be worth a HD, to step 4 might be an A, but step 2 is needed for a C.

The writeup could be much improved. Watch your spelling.
In the introduction, "This dissertation will investigate the concept of developing an automated vehicle guidance system using the Android platform" should appear before the background.

Appendix B Elementary Android code

Android developers ('Android Developers' 2015) has a list of training modules for application development. These developer training modules were used as a valuable resource to identify what certain aspects of Android code is used for and they provided worked examples of how Android developers should use code segments.

B.1. Android terms

From the elementary code that was developed an understanding of the following terms and classes was vital for further Android development. The first four terms are Android components.

- **Activities** – An activity is used for each new screen with a user interface. Activities can be shared between different Android applications if permission is granted.
- **Services** – A service is a background process that has no user interface and can only be accessed through another Android component such as an activity.
- **Content provider** – A content provider manages access and use of a data set that is shared between different applications.
- **Broadcast receiver** – A broadcast receiver responds to system wide announcements and takes the required actions such as a warning display when the battery is flat.
- **Permissions** – Every Android application must request and be granted permission from the android device prior to accessing android resources. This includes access and use of the camera, storage space, Bluetooth, GPS, and all other sensors.
- **Intents** - An intent sends asynchronous messages between components defining specific actions that need to be performed.
- **Manifest** – The manifest is an EXtensible Markup Language file under the name of AndroidManifest.xml, that holds a list of all the application components. Here components are defined as an activity, a service, a receiver, or a provider. The manifest also defines application permissions and requirements, a minimum API level, and it holds links to external libraries.
- **Thread** – A thread is a concurrent unit of execution used to run two segments of system code in parallel.
- **Callback** – A callback is an android method that will wait idle until it receives a callback notification from some other piece of code.
- **.apk** – An entire Android package that can be installed on an Android device.
- **Canvas** – a surface class that enables drawing.
- **Dalvik** – The name of the Android Virtual Machine

B.2. Camera control

The code below is one of the introductory camera classes that was built

```
/**
 * Created by shaun on 18/05/2015.
 */
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.Date;
import android.app.Activity;
import android.content.Context;
import android.content.pm.PackageManager;
import android.hardware.Camera;
import android.hardware.Camera.CameraInfo;
import android.hardware.Camera.PictureCallback;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.view.WindowManager;
import android.widget.Button;
import android.widget.LinearLayout;
import android.widget.Toast;

public class AndroidCameraExample extends Activity {
    private Camera mCamera;
    private CameraPreview mPreview;
    private PictureCallback mPicture;
    private Button capture, switchCamera;
    private Context myContext;
    private LinearLayout cameraPreview;
    private boolean cameraFront = false;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);
        myContext = this;
        initialize();
    }

    private int findFrontFacingCamera() {
        int cameraId = -1;
        // Search for the front facing camera
        int numberOfCameras = Camera.getNumberOfCameras();
        for (int i = 0; i < numberOfCameras; i++) {
            CameraInfo info = new CameraInfo();
            Camera.getCameraInfo(i, info);
            if (info.facing == CameraInfo.CAMERA_FACING_FRONT) {
                cameraId = i;
                cameraFront = true;
                break;
            }
        }
        return cameraId;
    }

    private int findBackFacingCamera() {
        int cameraId = -1;
        //Search for the back facing camera
        //get the number of cameras
        int numberOfCameras = Camera.getNumberOfCameras();
        //for every camera check
        for (int i = 0; i < numberOfCameras; i++) {
            CameraInfo info = new CameraInfo();
            Camera.getCameraInfo(i, info);
            if (info.facing == CameraInfo.CAMERA_FACING_BACK) {
                cameraId = i;
                cameraFront = false;
                break;
            }
        }
        return cameraId;
    }

    public void onResume() {
        super.onResume();
        if (!hasCamera(myContext)) {
            Toast toast = Toast.makeText(myContext, "Sorry, your phone does not have a camera!",
Toast.LENGTH_LONG);
            toast.show();
            finish();
        }
        if (mCamera == null) {
            //if the front facing camera does not exist
            if (findFrontFacingCamera()  1) {
                //release the old camera instance
                //switch camera, from the front and the back and vice versa

                releaseCamera();
                chooseCamera();
            } else {
                Toast toast = Toast.makeText(myContext, "Sorry, your phone has only one camera!",
Toast.LENGTH_LONG);
```

```
                toast.show();
            }
        }
    };

    public void chooseCamera() {
        //if the camera preview is the front
        if (cameraFront) {
            int cameraId = findBackFacingCamera();
            if (cameraId >= 0) {
                //open the backFacingCamera
                //set a picture callback
                //refresh the preview

                mCamera = Camera.open(cameraId);
                mPicture = getPictureCallback();
                mPreview.refreshCamera(mCamera);
            }
        } else {
            int cameraId = findFrontFacingCamera();
            if (cameraId >= 0) {
                //open the backFacingCamera
                //set a picture callback
                //refresh the preview

                mCamera = Camera.open(cameraId);
                mPicture = getPictureCallback();
                mPreview.refreshCamera(mCamera);
            }
        }
    }

    @Override
    protected void onPause() {
        super.onPause();
        //when on Pause, release camera in order to be used from other applications
        releaseCamera();
    }

    private boolean hasCamera(Context context) {
        //check if the device has camera
        if (context.getPackageManager().hasSystemFeature(PackageManager.FEATURE_CAMERA)) {
            return true;
        } else {
            return false;
        }
    }

    private PictureCallback getPictureCallback() {
        PictureCallback picture = new PictureCallback() {

            @Override
            public void onPictureTaken(byte[] data, Camera camera) {
                //make a new picture file
                File pictureFile = getOutputMediaFile();

                if (pictureFile == null) {
                    return;
                }
                try {
                    //write the file
                    FileOutputStream fos = new FileOutputStream(pictureFile);
                    fos.write(data);
                    fos.close();
                    Toast toast = Toast.makeText(myContext, "Picture saved: " + pictureFile.getName(),
Toast.LENGTH_LONG);
                    toast.show();

                } catch (FileNotFoundException e) {
                } catch (IOException e) {
                }

                //refresh camera to continue preview
                mPreview.refreshCamera(mCamera);
            }
        };
        return picture;
    }

    OnClickListener captrureListener = new OnClickListener() {
        @Override
        public void onClick(View v) {
            mCamera.takePicture(null, null, mPicture);
        }
    };

    //make picture and save to a folder
    private static File getOutputMediaFile() {
        //make a new file directory inside the "sdcard" folder
        File mediaStorageDir = new File("/sdcard/", "JCG Camera");

        //if this "JCGCamera folder does not exist
        if (!mediaStorageDir.exists()) {
            //if you cannot make this folder return
            if (!mediaStorageDir.mkdirs()) {
                return null;
            }
        }

        //take the current timeStamp
        String timeStamp = new SimpleDateFormat("yyyyMMdd_HHmmss").format(new Date());
        File mediaFile;
        //and make a media file:
        mediaFile = new File(mediaStorageDir.getPath() + File.separator + "IMG_" + timeStamp + ".jpg");

        return mediaFile;
    }

    private void releaseCamera() {
        // stop and release camera
        if (mCamera != null) {
            mCamera.release();
            mCamera = null;
        }
    }
}


        import java.io.IOException;

        import android.content.Context;
        import android.hardware.Camera;
        import android.util.Log;
        import android.view.SurfaceHolder;
        import android.view.SurfaceView;

public class CameraPreview extends SurfaceView implements SurfaceHolder.Callback {
    private SurfaceHolder mHolder;
    private Camera mCamera;

    public CameraPreview(Context context, Camera camera) {
        super(context);
        mCamera = camera;
        mHolder = getHolder();
        mHolder.addCallback(this);
        // deprecated setting, but required on Android versions prior to 3.0
        mHolder.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS);
    }

    public void surfaceCreated(SurfaceHolder holder) {
        try {
            // create the surface and start camera preview
            if (mCamera == null) {
                mCamera.setPreviewDisplay(holder);
                mCamera.startPreview();
            }
        } catch (IOException e) {
            Log.d(VIEW_LOG_TAG, "Error setting camera preview: " + e.getMessage());
        }
    }

    public void refreshCamera(Camera camera) {
        if (mHolder.getSurface() == null) {
            // preview surface does not exist
            return;
        }
        // stop preview before making changes
        try {
            mCamera.stopPreview();
        } catch (Exception e) {
            // ignore: tried to stop a non-existent preview
        }
        // set preview size and make any resize, rotate or
        // reformatting changes here
        // start preview with new settings
        setCamera(camera);
        try {
            mCamera.setPreviewDisplay(mHolder);
            mCamera.startPreview();
        } catch (Exception e) {
            Log.d(VIEW_LOG_TAG, "Error starting camera preview: " + e.getMessage());
        }
    }

    public void surfaceChanged(SurfaceHolder holder, int format, int w, int h) {
        // If your preview can change or rotate, take care of those events here.
        // Make sure to stop the preview before resizing or reformatting it.
        refreshCamera(mCamera);
    }

    public void setCamera(Camera camera) {
        //method to set a camera instance
        mCamera = camera;
    }

    @Override
    public void surfaceDestroyed(SurfaceHolder holder) {
        // TODO Auto-generated method stub
        // mCamera.release();

    }
```

B.3. GPS access example code

This code was an introductory look into accessing the GPS sensor.

```
package com.shaun.getgpslocation;

import android.app.AlertDialog;
import android.app.Service;
import android.content.Context;
import android.content.DialogInterface;
import android.content.Intent;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.os.IBinder;
import android.provider.Settings;

/**
 * Created by shaun on 10/05/2015.
 */
public class GPSTracker extends Service implements LocationListener {

    private final Context context;

    boolean isGPSEnabled = false;
    boolean isNetworkEnabled = false;
    boolean canGetLocation = false;

    Location location;

    double latitude;
    double longitude;

    private static final long MIN_DISTANCE_CHANGE_FOR_UPDATES = 1; // 1 METER
    private static final long MIN_TIME_BW_UPDATES = 1000 * 30 * 1; // 30 SECOND

    protected LocationManager locationManager;

    public GPSTracker(Context context) {
        this.context = context;
        getLocation();
    }

    public Location getLocation() {
        try {
            locationManager = (LocationManager) context.getSystemService(LOCATION_SERVICE);

            isGPSEnabled = locationManager.isProviderEnabled(LocationManager.GPS_PROVIDER);

            isNetworkEnabled = locationManager.isProviderEnabled(LocationManager.NETWORK_PROVIDER);

            if (!isGPSEnabled && !isNetworkEnabled){

            } else {
                this.canGetLocation = true;

                if (isNetworkEnabled) {
                    locationManager.requestLocationUpdates(LocationManager.NETWORK_PROVIDER, MIN_TIME_BW_UPDATES,
MIN_DISTANCE_CHANGE_FOR_UPDATES, this);


                    if (locationManager != null) {
                        location = locationManager.getLastKnownLocation(LocationManager.NETWORK_PROVIDER);

                        if (location != null) {
                            latitude = location.getLatitude();
                            longitude = location.getLongitude();
                        }
                    }

                }

                if (isGPSEnabled) {
                    if (location == null) {
                        locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER, MIN_TIME_BW_UPDATES,
MIN_DISTANCE_CHANGE_FOR_UPDATES, this);

                        if (locationManager != null) {
                            location = locationManager.getLastKnownLocation(LocationManager.GPS_PROVIDER);

                            if (location != null) {
                                latitude = location.getLatitude();
                                longitude = location.getLongitude();
                            }
                        }
                    }
                }
            }
        } catch (Exception e) {
            e.printStackTrace();
        }

        return location;
    }

    public void stopUsingGPS() {
        if (locationManager != null) {
            locationManager.removeUpdates(GPSTracker.this);
        }
    }

    public double getLatitude() {
        if (location != null) {
            latitude = location.getLatitude();
        }
        return latitude;
    }

    public double getLongitude() {
        if (location != null) {
            longitude = location.getLongitude();
        }
        return longitude;
    }

    public boolean isCanGetLocation() {
        return this.canGetLocation;
    }

    public void showSettingsAlert() {
        AlertDialog.Builder alertDialog = new AlertDialog.Builder(context);

        alertDialog.setTitle("GPS is setting");

        alertDialog.setMessage("GPS is not enabled. Do you want to go to the settings menu?");

        alertDialog.setPositiveButton("Settings", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                Intent intent = new Intent(Settings.ACTION_LOCATION_SOURCE_SETTINGS);
                context.startActivity(intent);
            }
        });

        alertDialog.setNegativeButton("Cancel", new DialogInterface.OnClickListener() {
            @Override
            public void onClick(DialogInterface dialog, int which) {
                dialog.cancel();
            }
        });
        alertDialog.show();
    }

    @Override
    public void onLocationChanged(Location location) {

    }

    @Override
    public void onStatusChanged(String provider, int status, Bundle extras) {

    }

    @Override
    public void onProviderEnabled(String provider) {

    }

    @Override
    public void onProviderDisabled(String provider) {

    }

    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }
}


package com.shaun.getgpslocation;

import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;


public class MainActivity extends ActionBarActivity {

    Button btnShowLocation;

    GPSTracker gps;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        btnShowLocation = (Button) findViewById(R.id.show_location);

        btnShowLocation.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                gps = new GPSTracker(MainActivity.this);

                if (gps.canGetLocation) {
                    double latitude = gps.getLatitude();
                    double longitude = gps.getLongitude();

                    Toast.makeText(getApplicationContext(), "Your Location is -\nLat: " + latitude + "\nLong: " +
longitude, Toast.LENGTH_LONG).show();
                } else {
                    gps.showSettingsAlert();
                }
            }
        });
    }
}
```

B.4. BoofCV LineDetection and colorSegment example code

Out of the many BoofCV example code segments looked at these Java classes provided great examples for me to investigate.

```
import java.awt.image.BufferedImage;

public class ExampleLineDetection {

    // adjusts edge threshold for identifying pixels belonging to a line
    private static final float edgeThreshold = 25;
    // adjust the maximum number of found lines in the image
    private static final int maxLines = 10;

    /**
     * Detects lines inside the image using different types of Hough detectors
     *
     * @param image Input image.
     * @param imageType Type of image processed by line detector.
     * @param derivType Type of image derivative.
     */
    public static<T extends ImageSingleBand, D extends ImageSingleBand>
    void detectLines( BufferedImage image ,
                      Class<T> imageType ,
                      Class<D> derivType )
    {
        // convert the line into a single band image
        T input = ConvertBufferedImage.convertFromSingle(image, null, imageType );

        // Comment/uncomment to try a different type of line detector
        DetectLineHoughPolar<T,D> detector = FactoryDetectLineAlgs.houghPolar(
                new ConfigHoughPolar(3, 30, 2, Math.PI / 180,edgeThreshold, maxLines), imageType, derivType);
//    DetectLineHoughFoot<T,D> detector = FactoryDetectLineAlgs.houghFoot(
//          new ConfigHoughFoot(3, 8, 5, edgeThreshold,maxLines), imageType, derivType);
//    DetectLineHoughFootSubimage<T,D> detector = FactoryDetectLineAlgs.houghFootSub(
//          new ConfigHoughFootSubimage(3, 8, 5, edgeThreshold,maxLines, 2, 2), imageType, derivType);

        List<LineParametric2D_F32> found = detector.detect(input);

        // display the results
        ImageLinePanel gui = new ImageLinePanel();
        gui.setBackground(image);
        gui.setLines(found);
        gui.setPreferredSize(new Dimension(image.getWidth(),image.getHeight()));

        ShowImages.showWindow(gui,"Found Lines");
    }

    /**
     * Detects segments inside the image
     *
     * @param image Input image.
     * @param imageType Type of image processed by line detector.
     * @param derivType Type of image derivative.
     */
    public static<T extends ImageSingleBand, D extends ImageSingleBand>
    void detectLineSegments( BufferedImage image ,
                             Class<T> imageType ,
                             Class<D> derivType )
    {
        // convert the line into a single band image
        T input = ConvertBufferedImage.convertFromSingle(image, null, imageType );

        // Comment/uncomment to try a different type of line detector
        DetectLineSegmentsGridRansac<T,D> detector = FactoryDetectLineAlgs.lineRansac(40, 30, 2.36, true,
imageType, derivType);

        List<LineSegment2D_F32> found = detector.detect(input);

        // display the results
        ImageLinePanel gui = new ImageLinePanel();
        gui.setBackground(image);
        gui.setLineSegments(found);
        gui.setPreferredSize(new Dimension(image.getWidth(),image.getHeight()));

        ShowImages.showWindow(gui,"Found Line Segments");
    }

    public static void main( String args[] ) {
        BufferedImage input = UtilImageIO.loadImage("../data/evaluation/simple_objects.jpg");

        detectLines(input,ImageUInt8.class,ImageSInt16.class);

        // line segment detection is still under development and only works for F32 images right now
        detectLineSegments(input, ImageFloat32.class, ImageFloat32.class);
    }
}
import java.awt.event.MouseAdapter;
import java.awt.image.BufferedImage;

/**
 * Example which demonstrates how color can be used to segment an image.  The color space is converted from RGB
into
 * HSV.  HSV separates intensity from color and allows you to search for a specific color based on two values
 * independent of lighting conditions.  Other color spaces are supported, such as YUV, XYZ, and LAB.
 *
 * @author Peter Abeles
 */
public class ExampleSegmentColor {

    /**
     * Shows a color image and allows the user to select a pixel, convert it to HSV, print
     * the HSV values, and calls the function below to display similar pixels.
     */
    public static void printClickedColor( final BufferedImage image ) {
        ImagePanel gui = new ImagePanel(image);
        gui.addMouseListener(new MouseAdapter() {
            @Override
            public void mouseClicked(MouseEvent e) {
                float[] color = new float[3];
                int rgb = image.getRGB(e.getX(),e.getY());
                ColorHsv.rgbToHsv((rgb >> 16) & 0xFF, (rgb >> 8) & 0xFF, rgb & 0xFF, color);
                System.out.println("H = " + color[0]+" S = "+color[1]+" V = "+color[2]);

                showSelectedColor("Selected",image,color[0],color[1]);
            }
        });

        ShowImages.showWindow(gui,"Color Selector");
    }

    /**
     * Selectively displays only pixels which have a similar hue and saturation values to what is provided.
     * This is intended to be a simple example of color based segmentation.  Color based segmentation can be done
     * in RGB color, but is more problematic due to it not being intensity invariant.  More robust techniques
     * can use Gaussian models instead of a uniform distribution, as is done below.
     */
    public static void showSelectedColor( String name , BufferedImage image , float hue , float saturation ) {
        MultiSpectral<ImageFloat32> input =
ConvertBufferedImage.convertFromMulti(image,null,true,ImageFloat32.class);
        MultiSpectral<ImageFloat32> hsv = input.createSameShape();

        // Convert into HSV
        ColorHsv.rgbToHsv_F32(input,hsv);

        // Euclidean distance squared threshold for deciding which pixels are members of the selected set
        float maxDist2 = 0.4f*0.4f;

        // Extract hue and saturation bands which are independent of intensity
        ImageFloat32 H = hsv.getBand(0);
        ImageFloat32 S = hsv.getBand(1);

        // Adjust the relative importance of Hue and Saturation.
        // Hue has a range of 0 to 2*PI and Saturation from 0 to 1.
        float adjustUnits = (float)(Math.PI/2.0);

        // step through each pixel and mark how close it is to the selected color
        BufferedImage output = new BufferedImage(input.width,input.height,BufferedImage.TYPE_INT_RGB);
        for( int y = 0; y < hsv.height; y++ ) {
            for( int x = 0; x < hsv.width; x++ ) {
                // Hue is an angle in radians, so simple subtraction doesn't work
                float dh = UtilAngle.dist(H.unsafe_get(x,y),hue);
                float ds = (S.unsafe_get(x,y)-saturation)*adjustUnits;

                // this distance measure is a bit naive, but good enough for to demonstrate the concept
                float dist2 = dh*dh + ds*ds;
                if( dist2 <= maxDist2 ) {
                    output.setRGB(x,y,image.getRGB(x,y));
                }
            }
        }

        ShowImages.showWindow(output,"Showing "+name);
    }

    public static void main( String args[] ) {
        BufferedImage image = UtilImageIO.loadImage("../data/applet/sunflowers.jpg");

        // Let the user select a color
        printClickedColor(image);
        // Display pre-selected colors
        showSelectedColor("Yellow",image,1f,1f);
        showSelectedColor("Green",image,1.5f,0.65f);
    }
}
import com.sun.javafx.iio.ImageStorage;

import java.awt.image.BufferedImage;

/**
 * Example demonstrating high level image segmentation interface.  An image segmented using this
 * interface will have each pixel assigned a unique label from 0 to N-1, where N is the number of regions.
 * All pixels which belong to the same region are connected.  These regions are also known as superpixels.
 *
 * @author Peter Abeles
 */
public class ExampleSegmentSuperpixels {

    /**
     * Segments and visualizes the image
     */
    public static <T extends ImageBase>
    void performSegmentation( ImageSuperpixels<T> alg , T color )
    {
        // Segmentation often works better after blurring the image.  Reduces high frequency image components
which
        // can cause over segmentation
        GBlurImageOps.gaussian(color, color, 0.5, -1, null);

        // Storage for segmented image.  Each pixel will be assigned a label from 0 to N-1, where N is the number
        // of segments in the image
        ImageSInt32 pixelToSegment = new ImageSInt32(color.width,color.height);

        // Segmentation magic happens here
        alg.segment(color,pixelToSegment);

        // Displays the results
        visualize(pixelToSegment,color,alg.getTotalSuperpixels());
    }

    /**
     * Visualizes results three ways.  1) Colorized segmented image where each region is given a random color.
     * 2) Each pixel is assigned the mean color through out the region. 3) Black pixels represent the border
     * between regions.
     */
    public static <T extends ImageBase>
    void visualize( ImageSInt32 pixelToRegion , T color , int numSegments  )
    {
        // Computes the mean color inside each region
        ImageType<T> type = color.getImageType();
        ComputeRegionMeanColor<T> colorize = FactorySegmentationAlg.regionMeanColor(type);

        FastQueue<float[]> segmentColor = new ColorQueue_F32(type.getNumBands());
        segmentColor.resize(numSegments);

        GrowQueue_I32 regionMemberCount = new GrowQueue_I32();
        regionMemberCount.resize(numSegments);

        ImageSegmentationOps.countRegionPixels(pixelToRegion, numSegments, regionMemberCount.data);
        colorize.process(color,pixelToRegion,regionMemberCount,segmentColor);

        // Draw each region using their average color
        BufferedImage outColor = VisualizeRegions.regionsColor(pixelToRegion,segmentColor,null);
        // Draw each region by assigning it a random color
        BufferedImage outSegments = VisualizeRegions.regions(pixelToRegion, numSegments, null);

        // Make region edges appear red
        BufferedImage outBorder = new BufferedImage(color.width,color.height,BufferedImage.TYPE_INT_RGB);
        ConvertBufferedImage.convertTo(color, outBorder, true);
        VisualizeRegions.regionBorders(pixelToRegion,0xFF0000,outBorder);

        // Show the visualization results
        ListDisplayPanel gui = new ListDisplayPanel();
        gui.addImage(outColor,"Color of Segments");
        gui.addImage(outBorder, "Region Borders");
        gui.addImage(outSegments, "Regions");
        ShowImages.showWindow(gui,"Superpixels", true);
    }

    public static void main(String[] args) {
        BufferedImage image = UtilImageIO.loadImage("../data/applet/segment/berkeley_horses.jpg");
//    BufferedImage image = UtilImageIO.loadImage("../data/applet/segment/berkeley_kangaroo.jpg");
//    BufferedImage image = UtilImageIO.loadImage("../data/applet/segment/berkeley_man.jpg");
//    BufferedImage image = UtilImageIO.loadImage("../data/applet/segment/mountain_pines_people.jpg");
//    BufferedImage image = UtilImageIO.loadImage("../data/applet/particles01.jpg");

        // Select input image type.  Some algorithms behave different depending on image type
        ImageType<MultiSpectral<ImageFloat32>> imageType = ImageStorage.ImageType.ms(3, ImageFloat32.class);
//    ImageType<MultiSpectral<ImageUInt8>> imageType = ImageType.ms(3,ImageUInt8.class);
//    ImageType<ImageFloat32> imageType = ImageType.single(ImageFloat32.class);
//    ImageType<ImageUInt8> imageType = ImageType.single(ImageUInt8.class);

//    ImageSuperpixels alg = FactoryImageSegmentation.meanShift(null, imageType);
//    ImageSuperpixels alg = FactoryImageSegmentation.slic(new ConfigSlic(400), imageType);
        ImageSuperpixels alg = FactoryImageSegmentation.fh04(new ConfigFh04(100,30), imageType);
//    ImageSuperpixels alg = FactoryImageSegmentation.watershed(null,imageType);

        // Convert image into BoofCV format
        ImageBase color = imageType.createImage(image.getWidth(),image.getHeight());
        ConvertBufferedImage.convertFrom(image, color, true);

        // Segment and display results
        performSegmentation(alg,color);
    }
}
```

Appendix C Row identification machine vision Android code

This appendix contains source code used to develop the machine vision application

C.1. Android manifest - AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="org.shaun.machinevision.android"
        android:versionCode="1"
        android:versionName="1.0">

   <uses-sdk android:minSdkVersion="10" android:targetSdkVersion="17" /> // Android version targets

   <uses-permission android:name="android.permission.CAMERA" /> // Request permission for Camera access
   <uses-feature android:name="android.hardware.camera" android:required="true" /> // a device with a camera is
required
   <uses-feature android:name="android.hardware.camera.autofocus" android:required="true" /> // camera autofocus
is required

   <application android:label="@string/app_name" android:icon="@drawable/ic_launcher"> // resources app_name in
string.xml and app icon in drawables
      <activity android:name="org.shaun.machinevision.android.Row_Follow_Main"
                android:screenOrientation="landscape"
                android:label="@string/app_name"> //Set main activity activities properties
         <intent-filter>
            <action android:name="android.intent.action.MAIN"/>
            <category android:name="android.intent.category.LAUNCHER"/>
         </intent-filter> //set the intent-filter properties for the main activity
      </activity>
   </application>
</manifest>
```

C.2. Main activity - Row_Follow_Main.java

```
/*
 * This file creates a cameraPreview object containing video information from the android device.  The format of
the
 * video is changed before being put through an algorithm that identifies crop rows using a small segment of
pixels
 * located in a ViewWindow segment of the cameraPreview image.  Image is then annotated with system data before
being
 * converted back to a Bitmap image that is displayed on the device screen.
 * .
 *
 * Author: Shaun Field
 * Date: 09/2015
 */


package org.shaun.machinevision.android; // Specify package name

import android.app.Activity;
import android.app.AlertDialog;
import android.app.AlertDialog.Builder;
import android.content.DialogInterface;
import android.content.DialogInterface.OnClickListener;
import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.graphics.Paint;
import android.graphics.Bitmap.Config;
import android.graphics.Paint.Style;
import android.graphics.Color;
import android.hardware.Camera;
import android.hardware.Camera.CameraInfo;
import android.hardware.Camera.Size;
import android.os.Bundle;
import android.view.SurfaceView;
import android.view.Window;
import android.widget.FrameLayout;
import java.util.ArrayList;
import java.util.List;

//External Libraries
import boofcv.android.ConvertBitmap; //Used for storage area and converts image to bitmap
import boofcv.android.ConvertNV21; // Converts cameraPreview NV21 to alternative formats
import boofcv.struct.image.ImageUInt8;//Used to create unsigned 8bit Images
import boofcv.struct.image.MultiSpectral;//Used to create multispectral images
/**
 * The Row_Follow_Main class takes a camera preview from an android device, performs some image processing, and
outputs
 * results onto the display.  The requested cameraPreview resolution of 320x240 at the default 30 fps
 */
```

```
public class Row_Follow_Main extends Activity implements Camera.PreviewCallback {

    //CONSTANTS
    static int CAMERA_WIDTH=320; // Sets 320 pixels as the Camera width
    static int CAMERA_HEIGHT=240; // Sets 240 pixels as the Camera height
    int VIEWPORT_HEIGHT=80; // Constant viewport height for algorithm calculations
    int VIEWPORT_WIDTH=30; // Constant viewport width for algorithm calculations
    double VIEWPORT_PLANT_PROPORTION = 0.4; // Constant percentage of plant in the viewport for adgustment of the
threshold level
    double X_TOLERANCE = 0.2; // Sets constant tolerance level based on percentage of screen width. used for
vanishingPoint and slope

    //Variables
    private Visualization mDraw; // Initialised in Constructor.  Visualisation is an inner class to allow
annotations on the video output
    private CameraPreview mPreview; // Initialised in Constructor. Creates the CameraPreview object to store the
camera video data
    private final Object lockOutput = new Object(); //For synchronization because two threads have access to the
same output canvas
    private Bitmap output; // bitmap used to output the image
    private Camera mCamera; // create the Camera Object assigned value during setupApp method

    private Canvas mCanvas; // Canvas for drawing on bitmap created in setupApp method
    private Paint windowPaint, fitLinePaint; // paint for windows and fitline created in setupApp method
    private MultiSpectral<ImageUInt8> specImg; // MultiSpectral image used to process videofeed image created in
setupApp method

    private ProcessingThread thread; //Image processing Thread created in setupApp method

    //Math variables used for processing calculations
    private int viewPortLeft, viewPortTop, viewPortRight, viewPortBottom; // viewPort boundaries
    private int windowCentreX; // window and view port centre x-coordinate value
    private int thresholdLevel; // holds value for the threshold level.
    private double quality; // holds quality value

    double fitmean, fitslope;
    private int good, bad;
    int horizon;
    private String  writeScreen, writeScreen2, writeScreen3;
    private boolean drawline;
    private byte[] processByte;
    private byte[] storage; //Storage area when converting between CameraPreview and formatting data type
    private final Object lockPic = new Object();// Synchronisation objects
    boolean flipHorizontal; //Sets orientation for camera

    // Constructor to request the window view and set layout.
    // Initialise Visualisation mDrae, CameraPreview mPreview
    // Create FrameLayout preview and add mDraw and mPreview objects
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        setContentView(R.layout.video);
        mDraw = new Visualization(this);
        mPreview = new CameraPreview(this,this,true);
        FrameLayout preview = (FrameLayout) findViewById(R.id.camera_preview);
        preview.addView(this.mPreview);
        preview.addView(this.mDraw);
    }

    //run SetUpApp method when onResume is called
    @Override
    protected void onResume() {
        super.onResume();
        this.setUpApp();
    }

    //all the actions needed when the camera is paused
    @Override
    protected void onPause() {
        super.onPause();
        if (mCamera != null){
            mPreview.setCamera(null);
            mCamera.setPreviewCallback(null);
            mCamera.stopPreview();
            mCamera.release();
            mCamera = null;
            thread.stopThread();
            thread = null;
        }
    }

    // Sets up the Camera and Init all other variables
    private void setUpApp() {

        //Setup camera and resolution
        mCamera = Camera.open(); //Selects and opens the back facing camera by default.
        Camera.Parameters param = mCamera.getParameters(); //Get camera parameters
        param.setPreviewSize(CAMERA_WIDTH, CAMERA_HEIGHT);// Set the resolution size to 320x240
        mCamera.setParameters(param); //apply the resolution settings

        //Setup the canvas and paints and drawline for annotation
        mCanvas = new Canvas();
        windowPaint = new Paint();
        windowPaint.setColor(Color.BLUE);
        windowPaint.setStyle(Style.STROKE);
        fitLinePaint = new Paint();
        fitLinePaint.setColor(Color.RED);
        fitLinePaint.setStyle(Style.STROKE);
        drawline =false;
        // Setup image variables
        // MultiSpectral holds a ImageUInt8 object for each R,G,B spectrum for each pixel in the image
        // Bitmap.createBitmap makes a Bitmap with 4 bytes per pixel capable of holding 4 8 bit channels. used
for RGB
        // ConvertBitmap.declareStorage creates a byte[] used to store the input Bitmap image
        specImg = new MultiSpectral<ImageUInt8>(ImageUInt8.class, CAMERA_WIDTH, CAMERA_HEIGHT,3); // a BoofCV
class To hold RGB image
        output = Bitmap.createBitmap(CAMERA_WIDTH, CAMERA_HEIGHT, Config.ARGB_8888); //Config.ARGB_8888=4 byte
Bitmap image
        storage = ConvertBitmap.declareStorage(output, storage); // a BoofCV class to create a byte array

        // Setup fixed viewPort VIEWPORT_HEIGHT x VIEWPORT_WIDTH pixels at the centre of the screen
        // Horizon is a horizontal line position 1/5 from the top of the screen
        // viewPortBottom is a horizontal line position 4/5 from the top of the screen
        // viewPortTop is a horizontal line position VIEWPORT_HEIGHT pixels above viewPortBottom
        horizon =(int) (CAMERA_HEIGHT*0.2); //Int used to calculate the vanishingPoint and viewport.  line on
bottom 20% of screen
        thresholdLevel = 128; // Sets the threshold level.  This currently doesn't change
        quality = 0; // Sets the starting quality to 0
        viewPortBottom = horizon *4; // the y-coordinate pixel value for the bottom of the ViewPort
        viewPortTop = viewPortBottom -VIEWPORT_HEIGHT; //the y-coordinate pixel value for the top of the ViewPort
        viewPortLeft = CAMERA_WIDTH/2-VIEWPORT_WIDTH/2; //the x-coordinate pixel value for the left of the
ViewPort
        viewPortRight = CAMERA_WIDTH/2+VIEWPORT_WIDTH/2;  //the x-coordinate pixel value for the right of the
ViewPort
        windowCentreX =CAMERA_WIDTH/2; //the x-coordinate pixel value for the viewPort and the entire screen

        //create and start the processing thread and start video feed
        thread = new ProcessingThread();
        thread.start(); // start image processing thread
        mPreview.setCamera(mCamera); //Start the video feed
    }

    //Method to process every new frame from the camera
    @Override
    public void onPreviewFrame(byte[] bytes, Camera camera) {
        synchronized (lockPic) {
            this.processByte = bytes;
        }
        thread.interrupt();
    }

    // Creates inner Visualization class containing an Activity variable.
    // This class creates the darwable window on the CameraPreview to allow annotations
    // The constructor sets the Activity variable.
    // setWillNotDraw method must also be set to false or the Draw method will not be called
    private class Visualization extends SurfaceView {
        Activity activity;
        public Visualization(Activity context ) {
            super(context);
            activity = context;
            setWillNotDraw(false);
        }

        //This method takes a canvas as an input alters the size and shape and positions it.
        //drawBitmap takes the output.bmp and places it's top left corner at co-ordinates 0,0 on the canvas.  The
null
        // refers to the Paint value which in this case is null
        //It must be synchronized because two threads have access to the output canvas
        @Override
        protected void onDraw(Canvas canvas){
            synchronized (lockOutput) {
                int w = canvas.getWidth();
                int h = canvas.getHeight();
                double scaleX = w/(double) output.getWidth();
                double scaleY = h/(double) output.getHeight();
                double scale = Math.min(scaleX,scaleY);
                double tranX = (w-scale* output.getWidth())/2;
                double tranY = (h-scale* output.getHeight())/2;
                canvas.translate((float)tranX,(float)tranY);
                canvas.scale((float)scale,(float)scale);
                canvas.drawBitmap(output,0,0,null);
            }
        }
    }

    //Image processing thresd
    public class ProcessingThread extends Thread {

        //Variables for this class
        volatile boolean stopRequested; //Holds the stop request flag
        volatile boolean running; //Holds the running thread flag
        private float lineStartX,lineStartY,lineEndX,lineEndY; // start and end x and y coordinates for drawing
regression line
        double VanishingPointx; //Vanishing Point variable
        double snew, slopeLimit, vanishPointLimit; // variables for slope and vanishing point boundaries
        double qualityMin; //Variables for minimum quality value
        double averagePlantPix;// holds plant density for viewport
        ArrayList<Integer> mxyList = new ArrayList<Integer>(); //List for holding plant coordinates
        MultiSpectral<ImageUInt8> viewport;
        // Stops thread method
        public void stopThread() {
            this.stopRequested = true;
            while(this.running) {
                thread.interrupt();
                Thread.yield();
            }
        }
        //limit method to return greatest value
        private double limit (double v, double bound) {
            if (v>bound) {
                return bound;
            } else {
                if (v<-bound) {
                    return -bound;
                } else {
                    return v;
                }
            }
        } // End limit class

        //sgn method for returning 1,-1 or 0
        private int sgn(double num) {
            if (num>0) {
                return 1;
            } else {
                if (num<0) {
                    return -1;
                } else {
                    return 0;
                }
            }
        } // End sgn class

        //fit function to calculate xfit and sfit corrections
        public void fit() {
            double pixelX,pixelY;
            double m; //total pixel count
            double mx,mxx; //mx=total horizontal moment about the view window centre line; mxx=the second moment
            double my, myy, mxy, mxxfit; //my=mx but verticle, myy=mxx but verticle
            double denom;//denominator variable
            double mxx0=0; //second moment for  viewport

            // initialise variables
            mx=0; my=0; myy=0; mxy=0; mxx=0; mxxfit=0;
            //calculate the second moment for the viewport
            for(int i=1;i<viewport.width/2;i++) {
                mxx0+=i^2*viewport.height;
            }
            mxx0=mxx0*2;

            m=mxyList.size()/2;//Total plant pixels
            //loop through pixels to collect data list is (x,y,x,y...etc)
            for(int i=0;i<mxyList.size();i+=2) {
                //extract first 2 x and y coords where pixelX, pixelY are pixel coordinates from the main image
green
                // pixels and not the viewport image coordinates
                // pixelX range is now -35 to +35
                pixelX=mxyList.get(i)-windowCentreX;
                pixelY=mxyList.get(i+1);
                // This calculates the linear regression correction data
                mx += pixelX; // add pixelX to mx
                mxx += pixelX * pixelX; // add pixelX^2 to mxx
                mxy += pixelX * pixelY; // add pixelX*pixelY to mxy
                my += pixelY; // add pixelY to my
                myy += pixelY * pixelY; // add pixelY^2 to myy
            } //end list of pixels for loop

            //Below are calculations for fitmean and fitslope
            denom = m*myy-my*my; // denominator to calculate fitmean and fitslope
            //make sure there are sufficient values to make the calculations
            if(denom>10 && m>20){
                fitmean =(mx*myy-mxy*my)/denom; //calculate fitmean
                fitslope =(m*mxy-mx*my)/denom; //calculate fitslope
                mxxfit=mxx+m* fitmean * fitmean +myy* fitslope * fitslope -2* fitmean *mx;
                mxxfit+= -2* fitslope *mxy+2* fitslope * fitmean *my; //calculate new second moment
                quality =mxxfit/mxx0; // compare old second moment to new second moment to get quality value
            } else {
                quality =0.1; // set quality to low
            }
        } // end of fit method

        //Assess viewwindow
        public void assess() {

            //Declare variables
            double fitmeanNew, fitslopeNew; // stores the new corection data
            double viewportCentrex,viewportSlope; //stores the Viewport centre x value and slope value
            double TempFitx, TempFitslope;
            good =0; //Counter to check for a good fit
            fitmeanNew=0;
            fitslopeNew=0;
            averagePlantPix =((mxyList.size())/(VIEWPORT_HEIGHT*VIEWPORT_WIDTH));//Calculates the plant density
for the viewport

            //Set up for one viewport in the centre.
            viewportSlope = 0; //because viewport is in the centre of the window the slope should be 0
            viewportCentrex = VanishingPointx + windowCentreX - horizon * viewportSlope;  //This is the x-
coordinate for viewport centre
            fit(); // run the fit method to check for the line of fit answers held in global variables fitmean,
fitslope, quality

            TempFitx = fitmean + viewportCentrex; //store TempFitx
            TempFitslope = fitslope + viewportSlope; //store TempFitslope
            //if quality value is > 4
            if (quality > qualityMin) {
                //X and Y calues used for drawing the regression line
                lineStartY =0;
                lineEndY = 240;
                lineStartX = (float) (TempFitx+ lineStartY *TempFitslope); //startx value = my+xfit
                lineEndX = (float) (TempFitx+ lineEndY *TempFitslope); //endx value = my+xfit
                good += 1;
                fitmeanNew += fitmean;
                fitslopeNew += fitslope;
                drawline =true;
            }
            //Threshold Adjustement
            if(averagePlantPix<VIEWPORT_PLANT_PROPORTION){
                thresholdLevel++;
            } else {
                thresholdLevel--;
            }
            //Add new correction data if fix is good
            if (good >0) {
                fitslopeNew= limit(fitslopeNew / 2, X_TOLERANCE);
                // the following are the corrections to be made
                fitmeanNew= limit(fitmeanNew / 2, 4);
                double newVanishPointx =fitmeanNew+ horizon *fitslopeNew+ VanishingPointx; //this is the new
vanishing point
                newVanishPointx = limit(newVanishPointx/2,4); //set newVanishPointx limit
                snew = fitslopeNew;
                if(Math.abs(newVanishPointx - (horizon - VIEWPORT_HEIGHT/2) * snew)< vanishPointLimit &&
Math.abs(fitslopeNew)< slopeLimit) {
                    bad =0;
                    VanishingPointx = newVanishPointx;
                }
            } else {
                bad++;
                if(bad >10) {
                    VanishingPointx = VanishingPointx *0.9;
                }
            }
        } //assess finished

        // run method to start
        @Override
        public void run() {
            //initialise variables
            running=true; // Flag to say this thread is running
            VanishingPointx =0; // Set the vanishing point to 0
            vanishPointLimit =X_TOLERANCE*CAMERA_WIDTH;//Sets the limit of the vanishing point to 0.2x320=
            slopeLimit =X_TOLERANCE;//sets the slope limit to 0.2
            qualityMin =4.0;// sets minimum quality value to 4
            while( !this.stopRequested) {

                // Sleeps thread until it is told to do some work
                synchronized ( Thread.currentThread() ) {
                    try {
                        this.wait();
                    } catch (InterruptedException ignored) {}
                }
                mxyList.clear();//Clear all previous pixel results for every new frame
                lineEndY = 0; // so values don't continually add to old values
                lineStartY = 0; // so values don't continually add to old values
                //sync locks and process image data
                synchronized (lockPic) {
                    // subwindow size 30x80
                    ConvertNV21.nv21ToMsRgb_U8(processByte, specImg.width, specImg.height, specImg);//Convert
processByte NV21 to RGB
                    viewport = specImg.subimage((int) viewPortLeft, (int) viewPortTop, (int) viewPortRight, (int)
viewPortBottom, null); // create viewPort subimage
                    //Below code loops through pixels in viewPort and checks for greenness to identify it as a
plant plant pixels are stored.
                    for (int y = 0; y < viewport.getHeight(); y++) {
                        for (int x = 0; x < viewport.getWidth(); x++) {
                            // A pixel is a plant if its Green band is > the threshold level. threshold=128;
                            // Plant pixels are stored in mxyList for further processing.
                            //Plant pixels are turned red and non-plant are turned blue
                            if (255-viewport.getBand(1).get(x, y) > thresholdLevel) {
                                viewport.getBand(0).set(x,y,0); //Set red to 0
                                viewport.getBand(1).set(x,y,0); //Set Green to 0
                                viewport.getBand(2).set(x,y,255); //Set Blue 255
                            } else {
                                mxyList.add(x+(int) viewPortLeft); //add x coordinate to list
                                mxyList.add(y+(int) viewPortTop); //add y coordinate to list
                                viewport.getBand(0).set(x,y,255); //Set Red to 255
                                viewport.getBand(1).set(x,y,0);//Set Green to 0
                                viewport.getBand(2).set(x,y,0);//Set Blue to 0
                            } // end if  green pixel
                        } // end x pixel loop for subwindow
                    } // end y pixel loop for subwindow

                    //put the viewport window through the row follow algorithm by calling the assess method
                    assess();

                    //3 decimal places conversion for displaying value for messages output to screen
                    fitmean = fitmean *1000;
                    fitmean = Math.round(fitmean);
                    fitmean = fitmean /1000;
                    fitslope = fitslope *1000;
                    fitslope = Math.round(fitslope);
                    fitslope = fitslope /1000;
                    quality = quality *1000;
                    quality = Math.round(quality);
                    quality = quality /1000;
                    VanishingPointx = VanishingPointx *1000;
                    VanishingPointx =Math.round(VanishingPointx);
                    VanishingPointx = VanishingPointx /1000;
                    //Messages to output to screen
                    //Standard output correction data
                    writeScreen = "X_alignment= "+ fitmean +"  Slope_alignment "+ fitslope;
                    writeScreen2 = "Quality= "+ quality;
                    writeScreen3 = "Vanishing_Point = "+ VanishingPointx;

                    /* Uncomment to //Print Threshold data to screen
                    writeScreen = "Plant Pixels = "+(mxyList.size())+"  Viewport pixels "+
VIEWPORT_HEIGHT*VIEWPORT_WIDTH;
                    writeScreen2 = "Plant Density = "+ averagePlantPix*100+"%";
                    writeScreen3 = "Threshold = "+ thresholdLevel;
                    */
                }

                // lock the output and write to output screen
                synchronized (lockOutput) {
                    ConvertBitmap.multiToBitmap(specImg, output, storage);// Converts specImg to output using
storage array
                    mCanvas.setBitmap(output);//Set frame Bitmap as Canvas background
                    //write text to screen
                    mCanvas.drawText(writeScreen, 5, 10, windowPaint);
                    mCanvas.drawText(writeScreen2, 5, 20, windowPaint);
                    mCanvas.drawText(writeScreen3, 5, 30, windowPaint);
                    //draw regression line if Quality is > 4
                    if(drawline) {
                        mCanvas.drawLine(lineStartX, lineStartY, lineEndX,lineEndY, fitLinePaint); //Draw
regression line on Canvas
                        drawline =false; //Set drawline flag to false
                    }
                    //draw viewPort outline onto canvas
                    mCanvas.drawRect(viewPortLeft, viewPortTop, viewPortRight, viewPortBottom, windowPaint);
                }
                //output Canvas to screen
                mDraw.postInvalidate(); // Called to update the GUI with the news display
            }
            this.running = false;
        }
    }
}
```

C.3. CameraPreview.java class

```
/*
 * This file creates accesses the android camera and makes the camera preview have the same resolution as the
camera
 * input resolution.  This class is built on reccomendations and program testing carried out by BoofCV.
 *
 * BoofCV (http://boofcv.org) is an open source Java machine vision library that is Licensed under the Apache
License
 * Version 2.0 avaliable from http://www.apache.org/licenses/LICENSE-2.0f
 *
 * Author: Shaun Field
 * Date: 09/2015
 */

package org.shaun.machinevision.android;

import android.content.Context;
import android.hardware.Camera;
import android.hardware.Camera.PreviewCallback;
import android.hardware.Camera.Size;
import android.util.Log;
import android.view.SurfaceHolder;
import android.view.SurfaceView;
import android.view.View;
import android.view.ViewGroup;
import static android.view.SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS;

public class CameraPreview extends ViewGroup implements SurfaceHolder.Callback {
   //Declare variable
   private final String logMessageString;
   SurfaceView mSurfaceView;
   SurfaceHolder mHolder;
   Camera mCamera;
   PreviewCallback previewCallback;
   boolean hidden;

   // The constructor for the CameraPreview object
   public CameraPreview(Context context, PreviewCallback previewCallback, boolean hidden ) {
      // make the CameraPreview variables equal the passed in variables
      super(context);
      logMessageString = "CameraPreview Initialised";
      this.previewCallback = previewCallback;
      this.hidden = hidden;
      mSurfaceView = new SurfaceView(context);
      addView(mSurfaceView);
      //Callback for create and destroy notifications
      mHolder = mSurfaceView.getHolder();
      mHolder.addCallback(this);
      mHolder.setType(SURFACE_TYPE_PUSH_BUFFERS);
   }

   //Create and set the camera instance
   public void setCamera(Camera camera) {
      mCamera = camera;
```

```
        // if there is no camera object then create one
        if (mCamera != null) {
            startPreview();
            requestLayout();
        }
    }

    //This method hides/unhides the video Preview and sets the size
    @Override
    protected void onMeasure(int widthIn, int heightIn) {
        int width,height;
        if(hidden) {
            width=height=2;
        } else {
            width = View.resolveSize(getSuggestedMinimumWidth(), widthIn);
            height = View.resolveSize(getSuggestedMinimumHeight(), heightIn);
        }
        setMeasuredDimension(width, height);
    }

    // Sets the cameraPreview layout
    @Override
    protected void onLayout(boolean changed, int left, int top, int right, int bottom) {
        if(this.mCamera == null )
            return;
        if (changed && this.getChildCount() > 0) {
            View child = this.getChildAt(0);
            int width = right - left;
            int height = bottom - top;
            Size size = this.mCamera.getParameters().getPreviewSize();
            int previewWidth = size.width;
            int previewHeight = size.height;
            // Center the layout
            if (width * previewHeight > height * previewWidth) {
                int scaledChildWidth = previewWidth * height / previewHeight;
                left = (width - scaledChildWidth) / 2;
                top = 0;
                right = (width + scaledChildWidth) / 2;
                bottom = height;
            } else {
                int scaledChildHeight = previewHeight * width / previewWidth;
                left = 0;
                top = (height - scaledChildHeight) / 2;
                right = width;
                bottom = (height + scaledChildHeight) / 2;
            }
            child.layout(left,top,right,bottom);
        }
    }

    //called to begin the CameraPreview
    protected void startPreview() {
        try {
            mCamera.setPreviewDisplay(this.mHolder);
            mCamera.setPreviewCallback(this.previewCallback);
            mCamera.startPreview();
        } catch (Exception e){
            Log.d(logMessageString, "Error starting camera preview: " + e.getMessage());
        }
    }

    //called to restart preview if there is a change in the Camera object
    @Override
    public void surfaceCreated(SurfaceHolder holder) {
        if (mCamera == null) {
            Log.d(logMessageString, "Camera is null.  Bug else where in code. ");
            return;
        }
        this.startPreview();
    }


    @Override
    public void surfaceDestroyed(SurfaceHolder holder) {}
    @Override
    public void surfaceChanged(SurfaceHolder holder, int format, int w, int h) {}
}
```

Appendix D Testing code

This section contains code segments used during application testing

D.1. Testing code segments.

The code in this Appendix shows some of the changes to the main program for testing purposes.

Test	Code

Video stream access	This code covers some of the Video Stream formats tested during development
	`private ImageUInt8 gray1,gray2;// 8 bit unsigned int`
	`private ImageSInt16 derivX,derivY; //16 bit signed int`
	`ImageFloat32 image = new ImageFloat32(100,150);// 32 bit float`
	`ConvertNV21.nv21ToMsYuv_U8(processByte,mspc1.width,mspc1.height,specImg);//YUV image`
	`ConvertNV21.nv21ToGray(bytes,gray1.width,gray1.height,gray1); //grey scale image`

Row Accuracy	This code covers the 5 test scenarios for Row Accuracy in section 6.5.1 . In order to get the pixels in the correct places the following code was input where the plant pixels are identified during the x,y, for loop of the viewport
	Scenario 1 Straight
	`if (x>10 || x<20) {`
	`    viewport.getBand(0).set(x,y,0); //Set red to 0`
	`    viewport.getBand(1).set(x,y,0); //Set Green to 0`
	`    viewport.getBand(2).set(x,y,255); //Set Blue 255`
	`} else {`
	`    mxyList.add(x+(int) viewPortLeft); //add x coordinate to list`
	`    mxyList.add(y+(int) viewPortTop); //add y coordinate to list`
	`    viewport.getBand(0).set(x,y,255); //Set Red to 255`
	`    viewport.getBand(1).set(x,y,0);//Set Green to 0`
	`    viewport.getBand(2).set(x,y,0);//Set Blue to 0`
	`} // end if  green pixel`
	Scenario 2 Right
	`if (x>0 || x<10) {`
	`    viewport.getBand(0).set(x,y,0); //Set red to 0`
	`    viewport.getBand(1).set(x,y,0); //Set Green to 0`
	`    viewport.getBand(2).set(x,y,255); //Set Blue 255`
	`} else {`
	`    mxyList.add(x+(int) viewPortLeft); //add x coordinate to list`
	`    mxyList.add(y+(int) viewPortTop); //add y coordinate to list`
	`    viewport.getBand(0).set(x,y,255); //Set Red to 255`
	`    viewport.getBand(1).set(x,y,0);//Set Green to 0`
	`    viewport.getBand(2).set(x,y,0);//Set Blue to 0`
	`} // end if  green pixel`
	
	Scenario 3 left
	`if (x>20 || x<10) {`
	`    viewport.getBand(0).set(x,y,0); //Set red to 0`
	`    viewport.getBand(1).set(x,y,0); //Set Green to 0`
	`    viewport.getBand(2).set(x,y,255); //Set Blue 255`
	`} else {`
	`    mxyList.add(x+(int) viewPortLeft); //add x coordinate to list`
	`    mxyList.add(y+(int) viewPortTop); //add y coordinate to list`
	`    viewport.getBand(0).set(x,y,255); //Set Red to 255`
	`    viewport.getBand(1).set(x,y,0);//Set Green to 0`
	`    viewport.getBand(2).set(x,y,0);//Set Blue to 0`
	`} // end if  green pixel`
	
	Scenario 4 clockwise
	`if (x=14 && y<40 || x=16 && y>40) {`
	`    mxyList.add(x+(int) viewPortLeft); //add x coordinate to list`
	`    mxyList.add(y+(int) viewPortTop); //add y coordinate to list`
	`    viewport.getBand(0).set(x,y,255); //Set Red to 255`
	`    viewport.getBand(1).set(x,y,0);//Set Green to 0`
	`    viewport.getBand(2).set(x,y,0);//Set Blue to 0`
	`} // end if  green pixel`

	Scenario 1 anti-clockwise
	`if (x=14 && y>40 || x=16 && y<40) {`
	`    mxyList.add(x+(int) viewPortLeft); //add x coordinate to list`
	`    mxyList.add(y+(int) viewPortTop); //add y coordinate to list`
	`    viewport.getBand(0).set(x,y,255); //Set Red to 255`
	`    viewport.getBand(1).set(x,y,0);//Set Green to 0`
	`    viewport.getBand(2).set(x,y,0);//Set Blue to 0`
	`} // end if  green pixel`

Processor Speed	This code covers the test code for the processor tests undertaken in section 6.5.2 for the Processor max, min, and average speeds
	This is the test Code for the test with a start-up delay
	This code is at the top of the threadProcess class
	`while( !this.stopRequested) {`
	`    long startTimer = System.currentTimeMillis();`
	And this code is at the bottom of the class
	`timerCount++;`
	`long stopTimer = System.currentTimeMillis();`
	
	`if(timerCount>150) {`
	`    timerCount2++;`
	`    totalProcessTime += stopTimer - startTimer;//Thread average timer`
	`    if ((stopTimer - startTimer) < minProcessTime)`
	`        minProcessTime = stopTimer - startTimer;`
	`    if ((stopTimer - startTimer) > maxProcessTime)`
	`        maxProcessTime = stopTimer - startTimer;`
	`    //Messages to output to screen`
	`    //Standard output correction data`
	`    writeScreen = "Process Time total = "+ totalProcessTime +",  Average = "+`
	`totalProcessTime/timerCount2;`
	`    writeScreen2 = "Max = "+ maxProcessTime;`
	`    writeScreen3 = "Min = "+ minProcessTime;`
	`} else {`
	`    //Messages to output to screen`
	`    //Standard output correction data`
	`    writeScreen = "Process Time total wait 5 second";`
	`    writeScreen2 = "Max = ";`
	`    writeScreen3 = "Min = ";`
	`}`