Autonomous Driving Robot

CS39440 Major Project Report

Author: Josh Wells ([jow102@aber.ac.uk](mailto:jow102@aber.ac.uk))

Supervisor: Patricia Shaw (phs@aber.ac.uk)

24th April 2022

Version 1.0 (Draft)

This report is submitted as partial fulfilment of a BSc degree in  
Artificial Intelligence & Robotics (GH76)

Department of Computer Science

Aberystwyth University

Aberystwyth

Ceredigion

SY23 3DB

Wales, UK

Declaration of originality

I confirm that:

* This submission is my own work, except where clearly indicated.
* I understand that there are severe penalties for Unacceptable Academic Practice, which can lead to loss of marks or even the withholding of a degree.
* I have read the regulations on Unacceptable Academic Practice from the University’s Academic Registry (AR) and the relevant sections of the current Student Handbook of the Department of Computer Science.
* In submitting this work, I understand and agree to abide by the University’s regulations governing these issues.

Name …………………………………………

Date ……………………………………………

Consent to share this work

By including my name below, I hereby agree to this project's report and technical work being made available to other students and academic staff of the Aberystwyth Computer Science Department.

Name …………………………………………

Date ……………………………………………

Acknowledgements

I am grateful to…

* Ben Weatherley (bew46@aber.ac.uk)
* Patricia Shaw (phs@aber.ac.uk)

Abstract

Increasingly cars are becoming more autonomous, giving warnings for speed limit signs and controlling the speed of the vehicle. Meanwhile, fully autonomous cars are being developed and tested on the roads in a variety of situations. There are many problems surrounding autonomous cars, and this project will start to explore some of those complexities.

The goal of this project was developing a Robot Operating System (ROS) based program to work with Turtlebot3, a relatively simple robot, inside a simulated environment (Gazebo) but with the potential for it to be transferred to a physical robot. The project aimed to tackle the issues surrounding lane detection, and the adjustment of the robot’s driving behavior to accurately follow the detected lane.

The main body of this project involved developing a computer vision technique using OpenCV, to achieve an accurate lane detection algorithm that is also fast enough to be ran on a live video feed with minimal delay. The end-goal of the project was the development of a robot that can autonomously navigate a simulated road environment while obeying UK traffic laws.

Contents

1. Background, Analysis & Process 6

1.1. Background 6

1.2. Analysis 7

1.3. Process 8

2. Design 9

2.1. Overall Architecture 9

2.2. Detailed Design 10

2.2.1. Support & Implementation Tools

2.2.2. Image Processing

2.2.3. Selection of Hough Lines

2.2.4. Calculating Center Line & Driving

3. Implementation 13

4. Testing 14

4.1. Overall Approach to Testing 14

4.2. Automated Testing 14

4.2.1. Unit Tests 14

4.2.2. User Interface Testing 14

4.2.3. Stress Testing 14

4.2.4. Other Types of Testing 14

4.3. Integration Testing 15

4.4. User Testing 15

5. Critical Evaluation 16

6. References 17

7. Appendices 18

A. Third-Party Code and Libraries 19

B. Ethics Submission Error! Bookmark not defined.

C. Code Samples 20

# Background, Analysis & Process

## Background

The goal of this project was developing a Robot Operating System (ROS) based program to work with Turtlebot3, a relatively simple robot, inside a simulated environment (Gazebo) but with the potential for it to be transferred to a physical robot. The project aimed to tackle the issues surrounding lane detection, and the adjustment of the robot’s driving behavior to accurately follow the detected lane.

The main body of this project involved developing a computer vision technique using OpenCV, to achieve an accurate lane detection algorithm that is also fast enough to be ran on a live video feed with minimal delay. The end-goal of the project was the development of a robot that can autonomously navigate a simulated road environment while obeying UK traffic laws.

I did not have any previous practical experience with implementing computer vision techniques, but I have an interest in computer vision and its applications. I very much wanted to learn how to successfully implement the techniques I had studied into a functioning program. The coupling of computer vision and robotics is what motivated me to select this autonomous driving robot as my project.

In preparation for this project, I taught myself methods for implementing OpenCV to work alongside ROS on a live, simulated robot inside a simulated environment. Mainly using a YouTube tutorial about the basics of OpenCV ***(Murtaza’s Workshop – Robotics and AI, 2020)***. I have plenty of practical experience programming in C++, so decided to continue the project using C++ over Python. I began by researching implementation techniques for OpenCV in C++, and created a few simple practice programs to help me get used to the OpenCV library.

In order to get OpenCV to work alongside ROS I had to make use of the CV\_BRIDGE ***(wiki.ros.org, 2017)*** library available for C++. This required some research and a small bit of practice to get the live raw image from the robot into a format which I could then use for image processing with OpenCV.

I assessed a number of different existing systems, including a paper entitled ‘Automatic Driving on Ill-Defined Roads: An Adaptive, Shape-constrained, Colour-based Method’ ***(Ososinski and Labrosse, 2013)***. This paper investigated the creation of a system that can identify the size and shape of the road ahead of it, before navigating along the road. The method discussed in this paper was effective, but complicated. Designed for use on ill-defined roads (e.g. country roads or lanes), this method adds multiple layers of complexity that my project did not require. As my project would take place inside a simulated, well-marked road environment there would be no need for the extra steps taken by this method to identify the drivable area in front of the robot.

Another method which I investigated made use of a series of filters, that are available within the OpenCV library, that would be able to detect the lines on either side of the road ***(Pysource, 2018)****.* This method works well when there are clear markings on the road, once the edges of the lane/road are detected I would then be able to successfully calculate the robots movement path along the center of the lane.

TALK ABOUT TESLA AUTO CARS, UBER CARS ETC. SENSORS THEY USE ETC

SECTION ON TURTLEBOT?

## Analysis

The problem involves being able to successfully detect the edges of a lane/road, from the background work I can conclude that there are many different ways that one can achieve this goal. The approach taken by Marek Ososinski and Frédéric Labrosse ***(2013)*** provides a robust solution to detecting the width of the road ahead of the robot. By making use of a shape-constrained, colour-based method they were able to determine the exact edges of the road and draw a trapezoidal shape based on the shape of the detected road edges. This shape would then be able to determine the drivable area in front of the robot and use this information to steer, keeping the robot in the center of the road at all times. While this approach is extremely robust and would provide an accurate lane following behavior for this project, it adds a number of layers of complexity that I felt would not be achievable in the time-span of this project.

In light of this, I felt a better method for this project would be to attempt to identify the marked lines on either side of the road before then calculating the centre of the lane for the robot to drive along. This approach, while not as robust as the previously mentioned one, seemed far more achievable in the time-span of this project.

The main tasks of the project were as follows:

1. Create a simulated road environment for the robot to drive in.
   * Develop road model.
   * Create launch files to open world, and spawn robot at same time.
2. Develop an algorithm that can successfully detect the markings on the road.
   * Investigate existing systems for inspiration.
3. Develop an algorithm to calculate where the edges of the lane are.
4. Calculate the centre of the detected lane.
5. Develop a lane following behaviour for the robot that will drive along the centre of the lane determined by the previous algorithm.

I arrived at this list of objectives based on what I believe is achievable in the time available for this project.

## Process

You need to describe briefly the life cycle model or research method that you used. You do not need to write about all of the different process models that you are aware of. Focus on the process model that you have used. It is possible that you needed to adapt an existing process model to suit your project; clearly identify what you used and how you adapted it for your needs.

The life cycle model that I used to develop my project was based on an Agile Scrum approach.

# Design

## Overall Architecture

The project has 3 major areas: image processing, selection of Hough Lines and the calculation of the centre of the lane and driving. The project makes use of a subscriber/publisher system using ROS. ROS acts as a middle man between the robot and the program, the program will subscribe and publish to ROS topics. These topics include the raw image from the robot, as well as the topic that controls the robots driving.

The image processing function takes the image from the robot and finds the edges of the lane using the white road markings. The detected road markings can then be used to decide the size of the lane. Using a selection algorithm, the program can decide the most appropriate points to use for lane detection.

Diagram

Description automatically generatedThe selection of the most appropriate Hough Lines/points can then be used to draw lines on the image to represent the edges of the lane. From these points it’s possible to calculate the centre of the lane detected ahead of the robot. Once the centre of the lane is calculated, a line can be drawn on the image to represent the path which the robot will be required to follow. Using the position of the this centre line on the image, the robot can decide if it needs to turn left/right or continue straight.

The flow diagram *(Fig1)* for this project is relatively simple, the robot should remain in a loop until it either reaches the end of the road or the program is manually shutdown.

Within this loop, the program should retrieve the image from the ROS topic, before passing the image through the image processing function. The image processing has multiple steps of its own that is not shown in this simplified diagram. After image processing is complete, the program will then select the most appropriate Hough Lines before calculating the centre of the lane ahead of the robot. From this information the program can decide where to steer the robot.

*Fig1 (above): Project flow diagram.*

## Detailed Design

### Support & Implementation Tools

The programming language selected for this project is C++; I chose this language as it provides an in-depth library that I feel gives me more control over the project that I do not get from other programming languages such as Python. Python in general is a fantastic choice for creating ROS programs, however I have far more practical experience working with C++ compared to Python. Due to this, C++ was the obvious choice for this project.

I made use of the CLion IDE to write the code, which gave me some basic debugging tools which helped to spot some of the more obvious errors before attempting to compile. I made use of Catkin to compile the ROS programs quickly, this would also provide me with fairly detailed error messages if any errors were found in the program which made turnaround times much faster when implementing new code.

This is a ROS based project, designed to run on a live robot using a subscriber/publisher system. ROS allows me to subscribe and publish to specific topics that then control the robot. The camera from the robot will publish its image to the ‘camera/rgb/image\_raw’ topic, my program then subscribes to that topic where it can then get a message containing the raw image direct form the robot. Using the cv\_bridge library, I am able to convert that message into an image that can then be used for image processing with OpenCV.

The robot is simulated within Gazebo, this allows me to do repeated tests with the program without needing a physical, real-world robot. Gazebo allowed me to create a test environment that mimicked a real-world road. Due to safety and other limitations with the project, a real-world robot operating on a real road was not possible. The environment model for the road was provided to me by Ben Weatherley, a student who completed a similar project last year entitled ‘Autonomous Driving Robot – Detecting and Reacting to Roadsigns’ ***(Weatherley, B., 2021)***. The road system modelled was that of a short loop, with a number of sharp 90 degree turns, 2 bus stops and 2 give-way junctions.

### Image Processing

Diagram

Description automatically generatedThe image processing is done in 5 steps; Cropping, Convert to HSV, Colour Mask, Edge Detection and finally Hough Line Generation. This process will result in a new edited image that has the edges of the lane marked and saved. This new image can then be used by the program for the later stages of the lane following behaviour.

The program will first retrieve the image from the ROS topic and convert it a useable OpenCV image ready for processing. This process will involve the use of the CV\_BRIDGE library, as the ROS topic that stores the raw image from the robot will only send that image to the program as a message which will need to be converted inside a subscriber call-back function.

The first step in the image processing technique for lane detection is to crop the image. The raw image from the robot will contain a lot of information that is not useful, and can actually cause extra issues if not removed. The upper half of the image should be cropped out to reduce processing time and complexity.

The image from the robot is in the Blue, Green, Red (BGR) colour space. This colour space provides a natural looking image and colours however it is not particularly easy to mask. To get around this, the image will be converted into the Hue, Saturation, Value (HSV) colour space. The HSV colour space is much simpler to mask specific colours, which will make the next step of the process much easier.

The image will then have a colour mask applied to it so that only areas that match the set range of HSV values will be seen on the image. In the case of my project the algorithm will mask off any area that is not white, masking everything but the road markings. Masking the unneeded areas from the image allows the Canny Edge Detection in the next step to be more accurate.

The image will then be ran through a Canny Edge Detection filter, which will create a number of points along the detected edges on the screen. The Canny Edge Detector provides much smoother edges than those produced by the Sobel Edge Detector at the expense of slightly more processing time.

The final step for the image processing will use the Hough Lines filter to generate lines on the image based off the points generated by the edge detector. These lines will each have their start and end coordinates stored in a vector, which will allow the next step of the program to sort the most appropriate points to use for lane tracking.

*Fig2 (above): Image Processing Flow Diagram*

### Selection of Appropriate Hough Lines

The generated Hough Lines are stored inside a vector, each line has 2 points (start and end) and each point has 2 values (X and Y coordinates). Selecting the most appropriate lines to use as markers for the edge of the lane requires an algorithm that can systematically go through each of the stored lines and decide if the position of the current line is better than the previously checked line.

Chart, line chart

Description automatically generatedFig3depicts an example image from the robot. The black lines on the image are the detected Hough Lines, representing the road markings visible to the robot. Each of these lines will have start and end XY coordinates that will need to be checked in order to find the best possible points for tracking the edges of the lane.

*Fig3 (above): Example of Hough Lines selection.*

The algorithm will select the following:

* The lowest point on the left side of the image: *(Fig3, CIRCLE1 (RED))*
* The lowest point on the right side of the image: *(Fig3, CIRCLE2 (GREEN))*
* The highest point on the left side of the image: *(Fig3, CIRCLE3 (MAGENTA))*
* The highest point on the right side of the image: *(Fig3, CIRCLE4 (BLUE))*

Chart, line chart

Description automatically generatedOnce the algorithm has selected the 4 most appropriate points, the program will draw a left and right line on the image between the relevant points, that will represent the edges of the lane.

Fig4 shows the result of the Hough Line selection process. The drawn lines represent the edges of the detected lane *(Fig4, LINE1 (BLUE), LINE2 (RED)).*

*Fig4 (above): Example of lane edge detection.*

### Calculating Centre of Detected Lane & Driving

Chart, line chart

Description automatically generatedThe centre of the lane is calculating using the results from the Hough Lines selection. The lines generated on the left and right side of the image are each made from the 2 relevant points selected by the algorithm. To draw the line in the centre of the lane, the program will calculate the centre of the top 2 points, as well as the centre of the bottom 2 points. This gives 2 new points that the program can then draw a line between, representing the calculated centre of the lane.

*Fig5 (above): Example of centre line calculation.*

Fig5 shows an example of the steps taken to calculate the centre of the lane *(Fig5, LINE2 (GREEN)).* The left and right lines are drawn from the most appropriate Hough Line points selected in the previous step *(Fig5, LINE1 (BLUE), LINE3 (RED))*. The arrows drawn between the points are shown for the purpose of this diagram, and are not actually drawn on the image from the robot. The centre points calculated by the algorithm are depicted by the circles at either end of the centre line *(Fig5, LINE2 (GREEN))*.

Each time the call back function updates the image from the robot, the image processing will repeat each of the steps mentioned in the above sections. When the centre line has been calculated, the driving function will make a decision based on the position of the line on the image.

An algorithm will calculate how far the centre green line is from the middle of the image, and if it is to the left or the right. From this, the program will decide if the robot needs to turn left/right, or continue straight. As the detected lane in front of the robot turns around corners, the centre line will skew to the right or left allowing the robot to figure out how much it needs to turn to remain in the centre of the lane. The program should always aim to have the centre of the lane in the centre of the image.

# Implementation

## Image Processing

### Failed Design

Before deciding to go with the above design, I explored a different method of tracking the lines on the road. This method made use of 2 image perspective warps, one for each side of the lane. This warp would make each side of the image, where the lines were expected to be, appear as if from a birds-eye view inside 2 separate cropped images. In theory, the line would appear in the centre of the warped image as a straight line from top to bottom. This process was meant to make the detection and tracking of the line easier, as the program would be able to drive based off the position of the line on the image. Due to the 2 lines being tracked separately from each other, it would be possible for the robot to drive relatively accurately when just one of the lines is in view/being tracked.

However, this design fell apart rather quickly. The warped image technique relied on the lines being in the exact same place every time and could not account for varying lane sizes. Other issues with this design included the program losing track of the line extremely easily, even the slightest of turns by the robot would cause the warped image to become almost entirely useless. When the robot would approach a corner, it would lose track of the line completely. I attempted to correct some of these issues by increasing the area of the image that gets warped, hoping that the larger size of the warped image would allow the line to remain visible in small turns. This fixed the problem for driving in a straight lane, but sharp corners remained un-driveable. Road markings such as the bus lanes/side of road parking and narrower sections of road would also cause this design to fail. After a week of trying to make this design function, I decided to abandon it and go back to the drawing board. This failed design made me realise that the problem of lane detection is much more complex that I originally assumed.

### Design Challenges

While implementing the successful design, I encountered a number of challenges. I overcame each of these challenges and produced a successful image processing technique for lane detection. Overall the implementation of the image processing was difficult, and I learned a lot about various filters and techniques while creating the program.

One of the first issues I encountered was with the application of a colour mask onto the image. The raw image from the robot is in the BGR colour space, which provides a natural coloured image that is great for viewing. However, BGR is not easy to apply a colour filter on, as it requires a more complicated set of filters. To get around this problem, I converted the image from BGR into HSV colour space. This colour space isn’t great for viewing, but the application of a colour mask is far simpler. To apply a mask to an HSV image, you only have to set an upper, and lower limit for the Hue, Saturation and Value. This can be done using 2 Scalar types, then inserting those Scalars into the OpenCV library function ‘inRange(imageIn, lower, upper, imageOut)’. The only parts of the image that are visible in the new image are the areas that had an HSV value between the upper and lower limits set in the function. For my program, everything but the white road markings would be masked off.

The 2nd challenge I encountered was tuning the parameters of the Canny Edge Detector ***(docs.opencv.org, n.d.)*** and Hough Lines functions ***(docs.opencv.org, n.d.)***. Running the program, noting the results of the functions and then adjusting the parameters before re-running the program was slow and hard to keep track of. I got around this by making use of Track Bars inside another image window. These Track Bars allowed me to assign a slider to a variable, which could be adjusted while the program is running. Doing this allowed me to run the program once, and adjust all of the parameters using the sliders and tune the functions in one go while also being able to see the live results of me changing the variables in the image windows output by the image processing function.

The Hough Lines required a lot more tuning than the Canny Edge Detector did. I repeatedly had to go back and adjust its parameters throughout the development of the project as I tested the program in various situations. I struggled to find the correct balance for the parameters, ensuring that it was sensitive enough to detect the smaller lines in the middle of the road, but not to sensitive that it mistook small errors in the edge detector as new lines. The settings that the Hough Lines function is currently using are good enough for the rough detection of the edges of the lane.

The image processing function was also far more intensive than I originally expected. My plan was to have the image processing done inside the call-back function for retrieving the raw image from the robot. However, this plan failed as soon as I tried to introduce a publisher object for the driving control. When ROS calls to update its call-back (subscriber) functions, they block other functions from being able to communicate with ROS. This meant that as the intensive and slow image processing was being completed, the driving function wouldn’t work. The image processing would also be required to loop as fast as possible so that the lane detection is up to date at all times, meaning as soon as the image processing finished it would instantly loop again which gave no time for the driving function to operate.

To get around this issue, I moved the image processing into its own function which would be called in turn. This meant that the driving function should always get a chance to publish to ROS between loops of the image processing. This solution worked for a short time, but as the image processing got more complex it eventually became obvious that a single thread system was not going to work. I then moved the image processing into its own separate thread, and created some global variables to allow cross-thread communication. This solution presented some small issues at first to do with OpenCV exceptions but these issues were quickly resolved by adding in some artificial slow-downs into the loop as well as giving the program some buffer time when it first opens (Waiting around 3 seconds for the program to properly start before attempting any image processing).

## Selection of Appropriate Hough Lines

### Failed Designs

Developing the algorithm to select the most appropriate Hough Lines from the vector of lines created by the image processing function was one of the most challenging parts of this project. This algorithm was required to sort through all of the generated lines, and calculate which of the saved points within each of those lines was the most accurate in order to track the edges of the lane. The algorithm became extremely confusing, and required a number of iterations before it finally produced a useable and semi-reliable outcome.

The first method I implemented would sieve through the vector of stored lines, and choose the 2 lines (left & right) that best fit. This technique did not predict the edges of the lane correctly, and would often get confused by other markings on the road or select lines that were produced due to a small error caused by poor parameter tuning in the Hough Lines function during the image processing. This method would only take into account whole lines as generated by the Hough Lines function, these lines would often be incomplete or would not extend the whole length of the lane edge.

The second approach I attempted would try to find the 2 highest points on the image, and then use those for tracking. The lower 2 points were fixed on the image. This presented some issues with tracking accuracy, the calculated edges of the lane were often incorrect and the algorithm would sometimes choose overlapping points.

The third design used an approach similar to the one in the final program, but lacked some crucial checks, and a ‘deadzone’ to reduce sensitivity. Due to this, the third design would often pick overlapping points which would then confuse the driving function.

The algorithm for selecting the Hough Lines was written, and re-written many times. Due to the confusing design, many more designs were developed to tackle the issue. Most failed designs never made it past a few lines of simple pseudocode before they were tossed in the bin. This ordeal made me realise that the problems of lane tracking would take me longer than expected to solve.

### Design Challenges

The algorithm became very complicated very quickly, and required many hours of testing different approaches before I landed on the current working method. The current algorithm is not perfect or very efficient, and has room for improvement. However, I ran out of time in this project and could not fix all of its problems. I wrote the algorithm’s pseudocode out many times on a piece of paper as a way of visualising the logic. The Hough Lines would be generated from left to right, and so the algorithm would need to be able to identify if the line is on the left or right side of the image, as well as which direction it is facing based off of the start and end points stored for each line. This meant the algorithm would need to be able to flip the lines start and end points so that they would be drawn from the bottom to the top of the image, instead of left to right, which would make the selection of the most appropriate points simpler to figure out.

The current algorithm had issues when approaching sharp turns in the road, where it would lose track of one side of the lane entirely. This was a massive issue that required a number of redundancies to be programmed so that the robot may continue driving until such time that the program is able to re-acquire the line. The addition of 2 bool types, that would check if the program has managed to find both left points, and both right points helped to reduce the number of false readings being given to the driving function. If the program was not able to detect 2 points, it would set the coordinates to a default number that the program would never reach naturally. The program performs 2 checks on the coordinates, one for each side of the lane. If it was not able to draw the left or right lane then the relevant bool is set to false. The driving function can then see the false reading and decide how to continue instead of receiving the wrong information as previous designs gave it. This reduced the number of errors when driving and improved the overall accuracy of the robot’s driving especially when going around sharp corners.

## Driving

### Design Challenges

The implementation of the driving function was the most straight forward part of this project, however it was not without its own issues. The first implementation of the driving failed to function altogether. This was due to an unforeseen issue with the image processing which caused it to block communication with ROS. This was not the main issue with the design, however. The main issue was that the driving function (when working) would cause the robot to jitter left and right very quickly and not actually go anywhere.

This issue was caused by a lack of a ‘deadzone’ when calculating whether the robot should turn left or right, plus the driving/turning speeds had not been correctly set and were far too fast. If the centre line was not perfectly in the centre of the screen, on the exact centre pixel, the robot would try to turn in order to correct this. Which caused the robot to be constantly fighting to have the line in the exact centre, which is not possible.

This introduction of a deadzone to the driving function meant that the line had a larger zone to be in that the program would then consider to be centred. The next issue caused by this was trying to tune the size of the deadzone so that it was big enough so the robot could drive forward smoothly, but making sure it wasn’t so big that the robot would drift too far out of the centre of the lane.

I also implemented the previously mentioned bool types that read true or false depending on if the relevant line was able to be drawn by the Hough Line selection process. For example, if the left line bool reads false then the program will apply a small left turn to the robot in an attempt to regain tracking of the left hand side line. The same process is used to regain tracking of the right hand side line if the program loses track of it. This driving function only changed a small amount from when it was first designed and implemented, it mostly operated without problem.

## Conclusion

In conclusion to the implementation section, I can compare the final project to the original requirements that are mentioned in the analysis section. The project has met each of the originally set requirements. It can successfully detect the left and right edges of the lane ahead of the robot, calculate the centre of the lane and then navigate the robot around the simulated road environment.

1. Create a simulated road environment for the robot to drive in.
   * Develop road model.
   * Create launch files to open world, and spawn robot at same time.
2. Develop an algorithm that can successfully detect the markings on the road.
   * Investigate existing systems for inspiration.
3. Develop an algorithm to calculate where the edges of the lane are.
4. Calculate the centre of the detected lane.
5. Develop a lane following behaviour for the robot that will drive along the centre of the lane determined by the previous algorithm.

The above requirements are what was set in the analysis phase of the project, and I believe the project has reached all of these targets successfully.

# Testing

Detailed descriptions of every test case are definitely not what is required in this section; the place for detailed lists of tests cases is in an appendix. In this section, it is more important to show that you adopted a sensible strategy that was, in principle, capable of testing the system adequately even if you did not have the time to test the system fully.

Provide information in the body of your report and the appendix to explain the testing that has been performed. How does this testing address the requirements and design for the project?

How comprehensive is the testing within the constraints of the project? Are you testing the normal working behaviour? Are you testing the exceptional behaviour, e.g. error conditions? Are you testing security issues if they are relevant for your project?

Have you tested your system on “real users”? For example, if your system is supposed to solve a problem for a business, then it would be appropriate to present your approach to involve the users in the testing process and to record the results that you obtained. Depending on the level of detail, it is likely that you would put any detailed results in an appendix.

Whilst testing with “real users” can be useful, don't see it as a way to shortcut detailed testing of your own. Think about issues discussed in the lectures about until testing, integration testing, etc. User testing without sensible testing of your own is not a useful activity.

The following sections indicate some areas you might include. Other sections may be more appropriate to your project.

## Overall Approach to Testing

ITERATIVE TESTING APPROACH

Subscribe to real world model/odometry topic for pos info. Or call service that does same thing then I can log xy coordinates.

## Automated Testing

### Unit Tests

### User Interface Testing

### Stress Testing

### Other Types of Testing

## Integration Testing

## User Testing

# Critical Evaluation

Examiners expect to find a section addressing questions such as:

* Were the requirements correctly identified?
* Were the design decisions correct?
* Could a more suitable set of tools have been chosen?
* How well did the software meet the needs of those who were expecting to use it?
* How well were any other project aims achieved?
* If you were starting again, what would you do differently?

Other questions can be addressed as appropriate for a project.

The questions are an indication of issues you should consider. They are not intended as a specification of a list of sections.

The evaluation is regarded as an important part of the project report; it should demonstrate that you are capable not only of carrying out a piece of work but also of thinking critically about how you did it and how you might have done it better. This is seen as an important part of an honours degree.

There will be good things in the work and aspects of the work that could be improved. As you write this section, identify and discuss the parts of the work that went well and also consider ways in which the work could be improved.

In the latter stages of the module, we will discuss the evaluation. That will probably be around week 9, although that differs each year.

# References

1. docs.opencv.org. (n.d.).   
   *OpenCV: Hough Line Transform*. [online] Available at: <https://docs.opencv.org/3.4/d9/db0/tutorial_hough_lines.html>.  
   [Accessed 5 March 2022]
2. docs.opencv.org. (n.d.).   
   *OpenCV: Canny Edge Detector*. [online] Available at: <https://docs.opencv.org/3.4/da/d5c/tutorial_canny_detector.html>.  
   [Accessed 5 March 2022]
3. Weatherley, B. (2021).   
   *Autonomous Driving Robot - Detecting and Reacting to Roadsigns*. Supplied 3D models to use in simulated environment.
4. PySource. (2018).   
   *Lines detection with Hough Transform – OpenCV 3.4 with python 3 Tutorial 21*. [online] Available at: <https://www.youtube.com/watch?v=KEYzUP7-kkU>  
   [Accessed 5 March 2022].
5. Ososinski, M. and Labrosse, F. (2013).   
   Automatic Driving on Ill-defined Roads: An Adaptive, Shape-constrained, Color-based Method. *Journal of Field Robotics*, 32(4), pp.504–533. doi:10.1002/rob.21494.
6. wiki.ros.org. (2017).   
   *cv\_bridge/Tutorials/UsingCvBridgeToConvertBetweenROSImagesAndOpenCVImages - ROS Wiki*. [online] Available at: <http://wiki.ros.org/cv_bridge/Tutorials/UsingCvBridgeToConvertBetweenROSImagesAndOpenCVImages>.  
   [Accessed 2March 2022]
7. Murtaza’s Workshop – Robotics & AI. (2020).   
   *LEARN OPENCV C++ in 4 HOURS | Including 3x Example Projects Win/Mac (2021)*. *YouTube*. Available at: <https://www.youtube.com/watch?v=2FYm3GOonhk>.  
   [Accessed 4 March 2022]

# Appendices

The appendices are for additional content that is useful to support the discussion in the report. It is material that is not necessarily needed in the body of the report, but its inclusion in the appendices makes it easy to access.

If you have used any 3rd party code, i.e. code that you have not written yourself such as libraries, then you must include Appendix A. In that appendix, you will provide details of the 3rd party code that you have used.

For most other items, it would be better to include them in your technical submission instead of including them as an appendix. For example:

* If you have developed a Design Specification document as part of a plan-driven approach for the project, then it would be appropriate to include that document in the technical work. In this report, you would highlight the most interesting aspects of the design, referring your reader to the full specification for further detail.
* If you have taken an agile approach to developing the project, then you may be less likely to have developed a full requirements specification at the start of the project. Perhaps you used stories to keep track of the functionality and the ‘future conversations.’ If it isn’t relevant to include all those stories in the body of your report, you could detail those stores in a document in the technical work.
* If you have used manual testing, then include a document in the technical work that records the tests that have been done. In this report, you would talk about the use of those tests.

Documents included in the technical work or in the appendices are supporting evidence of the work done. Where you include documents, this report should refer to the documents. You should not be relying on detailed study of those documents in order to understand what is written in this report.

Speak to your supervisor or the module coordinator if you have questions about this.

* 1. Third-Party Code and Libraries

If you have made use of any third-party code or software libraries, i.e. any code that you have not designed and written yourself, then you must include this appendix.

As has been said in lectures, it is acceptable and likely that you will make use of third-party code and software libraries. If third-party code or libraries are used, your work will build on that to produce notable new work. The key requirement is that we understand what your original work is and what work is based on that of other people.

Therefore, you need to clearly state what you have used and where the original material can be found. Also, if you have made any changes to the original versions, you must explain what you have changed.

The following is an example of what you might say.

**Apache POI library** – The project has been used to read and write Microsoft Excel files (XLS) as part of the interaction with the client’s existing system for processing data. Version 3.10-FINAL was used. The library is open source and it is available from the Apache Software Foundation ‎[5]. The library is released using the Apache License ‎[6]. This library was used without modification.

Include as many declarations as appropriate for your work. The specific wording is less important than the fact that you are declaring the relevant work.

* 1. Code Samples

This is an example appendix. Include as many appendices as you need. The appendices do not count towards the overall word count for the report.

For some projects, it might be relevant to include some code extracts in an appendix. You are not expected to put all of your code here - the correct place for all of your code is in the technical submission that is made in addition to the Project Report. However, if there are some notable aspects of the code that you discuss, including that in an appendix might be useful to make it easier for your readers to access.

As a general guide, if you are discussing short extracts of code then you are advised to include such code in the body of the report. If there is a longer extract that is relevant, then you might include it as shown in the following section.

Only include code in the appendix if that code is discussed and referred to in the body of the report.

Random Number Generator

The Bayes Durham Shuffle ensures that the pseudo random numbers used in the simulation are further shuffled, ensuring minimal correlation between subsequent random outputs.

// Some example code here…