



Prifysgol Cymru
Y Drindod Dewi Sant
University of Wales
Trinity Saint David

Efficient Carpark Utilisation via Image Processing

By

Rania Abdalla – 2222148

Supervisor – Dr Tim Bashford

Project submitted as part of the requirements for the
award of

MSc Software Engineering (Web Development)

January 2024

Abstract

A proof-of-concept application was developed to speed the time spent looking for a parking space in car parks. The goal of creating a scalable and efficient system to provide real-time updates led to the utilisation of computer vision and image processing techniques to detect parking spaces, determine their occupancy status, and inform users of their availability in real-time. Using Feature Driven Development (FDD) the prototype was developed in Python using the Flask API Framework and the OpenCV library for image processing. The results collected at the final stage of the prototype determined an average of 98.7% success rate for parking space detection, and 59.7% for occupancy status. These results showed that the prototype was partially successful and gave a clear vision for future work and improvements.

Table of Contents

1.	LIST OF FIGURES	4
2.	LIST OF TABLES	4
3.	INTRODUCTION	6
4.	LITERATURE REVIEW	7
	COMPUTER VISION	7
	<i>Introduction</i>	7
	<i>What is computer Vision?</i>	7
	<i>Image Processing Techniques</i>	12
	IMPLEMENTATION FRAMEWORKS AND LIBRARIES	19
	<i>Backend</i>	19
	<i>Frontend</i>	21
5.	EXISTING SOLUTIONS	22
	<i>Counter-based Systems</i>	22
	<i>Sensor-based Systems</i>	22
	<i>Vision-based systems</i>	22
6.	METHODOLOGY	23
	<i>Literature Review</i>	23
	<i>Software Development</i>	24
	<i>Result Analysis</i>	24
7.	DESIGN.....	25
	FEATURES	26
	<i>Feature 2: Occupancy Status</i>	26
	<i>Feature 3: Marking Bay as Occupied or Unoccupied</i>	27
	<i>Feature 4: RESTful Webservice</i>	27
8.	DATA ANALYSIS	27
9.	IMPLEMENTATION	29
	<i>Bay Segmentation/Detection</i>	29
	<i>Determining Space Status</i>	31
	<i>RESTful Service</i>	32
10.	RESULTS AND DISCUSSIONS.....	33
	<i>Summary</i>	35
11.	CONCLUSIONS AND RECOMMENDATIONS.....	37
	<i>Future Work</i>	37
	<i>Reflections</i>	38
12.	APPENDICES	39

1. List of figures

Figure 1.0 How computers view images [3]

Figure 2.0 Process of an Artificial Neural Network [11]

Figure 3.0: Convolutional Neural Network Diagram [13]

Figure 4.0: Example of how powerful Median Blur is at reducing noise [30].

Figure 5.0: Kernel for detecting edges along: x-direction (left) and y-direction (right) [33]

Figure 6.0: Edges detected separately around the x-direction and right direction [34]

Figure 7.0: Binary image with edges in white and black everything else [34]

Figure 8.0: Diagram showing hysteresis thresholding A is a strong edge pixel, B is a weak edge pixel, C is in between but is connected to A therefore kept as valid edge [38].

Figure 9.1: Morphological dilation process of expanding the shape of objects [43]

Figure 9.2: Morphological erosion process of shrinking the shape of an object [44]

Figure 10.0: Diagram illustrating intersection over union (IoU) formula [49]

Figure 11.0: Prototype design for the smart car parking system.

Figure 12.0: Car park image to be used for application highlighting potential drawbacks hinders.

Figure 13.0: Detected edges using Canny edge detector.

Figure 14.0: Image illustrating inner and outer edges of bay and some edges inside inner edge.

Figure 15.0: Some edges outside the bays with significant size difference.

Figure 16.0: Converting a Python object (list in this case) into a character stream in Pickle [59].

Figure 17.0: Results of detecting spaces in new car park images at various angles

Figure 18.0: Results of occupancy detection in new car park images at various angles

Figure 19.0: Target vs Actual results for space and occupancy detection

2. List of Tables

List of Tables

Table 1.0: Results of space and occupancy detection of car park images at various angles using count zero pixels method

Table 2.0: Results of space and occupancy detection of car park images at various angles using HSV segmentation method

Abstract

A proof-of-concept application was developed to speed the time spent looking for a parking space in car parks. The goal of creating a scalable and efficient system to provide real-time updates led to the utilisation of computer vision and image processing techniques to detect parking spaces, determine their occupancy status, and inform users of their availability in real-time. Using Feature Driven Development (FDD) the prototype was developed in Python using the Flask API Framework and the OpenCV library for image processing. The result collected at the final stage of the prototype determined an average of 98.7% success rate for parking space detection, and 59.7% for occupancy detection. These results showed that the prototype was partially successful and gave a clear vision for future work and improvements.

3. Introduction

Remote working became the norm during the pandemic, almost making us forget how frustrating it can be to find an available parking space at the beginning of the day. Finding a parking space at the DVLA can be quite a challenge. Although not everyone is on-site daily post pandemic, arriving after 08:30am on certain days often means spending 15-20 minutes searching for a parking spot. This can be particularly difficult in situations out of one's control.

To make matters worse, one of the DVLA car parks is a multi-storey with a one-way system. If you do not find a space on one level, you have to exit the car park, leave the DVLA grounds onto the main road, and then re-enter the multi-storey to try another level. This process can be incredibly frustrating, particularly at the hectic start of the day. As we gradually return to the office, and the parking issue has resurfaced, the demand for a solution is higher than ever.

While the inspiration for this project came from the DVLA's parking situation, it is important to note that car parking issues are a universal problem. [1] conducted a survey of over 100,000 drivers regarding their parking experiences, revealing that over 25% spend more than 15 minutes daily searching for a spot. 80% raised concerns about not knowing whether there are any available spaces, and 41% complained about insufficient parking spaces in general.

This research project aims to address this problem using modern technology. [2] conducted a study on three types of systems: counter-based, sensor-based, and vision based. Counter-based systems use sensors at the entrance or exit, counting cars and updating users on a screen without directing them to available spaces. Sensor-based systems, while more direct, can be costly due to the need for sensors and screens for each space. Vision-based systems are considered cost-effective and scalable, as they can be implemented using existing CCTV data to detect and update users on parking space availability.

The literature review of this paper explores various computer vision and image processing tools and techniques for implementing a vision-based car parking system. The gathered information will assist in developing a prototype application, featuring real-time detection and status updates for each car park.

In the final project stage, performance analysis will focus on achieving a prototype with 100% space detection accuracy for the utilised data and approximately 90% accuracy for new data. For occupancy detection, the goal is at least 80% accuracy for the used data and 70% accuracy for new data. The results will determine the future features and improvements required for a fully functional and scalable application adaptable to any car park.

If successful, this project has the potential to help eliminate the frustrations that come with finding an available parking space and for the DVLA specifically enhance employee satisfaction, productivity, and overall wellbeing.

4. Literature Review

The purpose of this literature review is to gain a deep understanding of Computer Vision and Image Processing to be able to design and implement a smart car parking system that can inform users of available parking spaces in real-time.

Computer Vision

Introduction

Computer vision enables a computer to see and understand the content of a digital image. It involves the development of algorithms and techniques that allow computers to recognise and process images just like the human eye. The human eye is incredible at deciphering objects, regardless of their distance or movement. Humans have the ability to instantly identify and determine the nature of objects in the real world and in images. Computer vision aims to train machines to replicate these actions, and this involves a few tasks.

What is computer Vision?

Computer vision is a field of Artificial Intelligence (AI), that involves training machines to perceive and interpret visual information similar to humans. This is achieved through a combination of software and hardware. In principle, computer vision consists of three key steps: image acquisition, image processing, and understanding the image. Despite its apparent simplicity, teaching a computer to recognise and interpret images is an intricate process that involves different algorithms and data manipulation techniques.

How Computers See Images

Looking at the left image in figure 1.0, as humans we can effortlessly see a few objects, a man using a camera mounted on a tripod likely capturing a scene within a field, there are some buildings in the background, and we also have the ability to differentiate the sky from the other objects in the image. A single glance allows us to make these distinctions, with the potential to identify additional objects upon closer examination. This recognition of objects is not as straight forward for computers [4]. Computers perceive images as numerical grids, as illustrated in the right image of figure 1.0. Each square in the grid represents a pixel which is the smallest unit of a digital image. The total number of pixels in an image is derived by multiplying the width and height values, often indicating the resolution of the image.

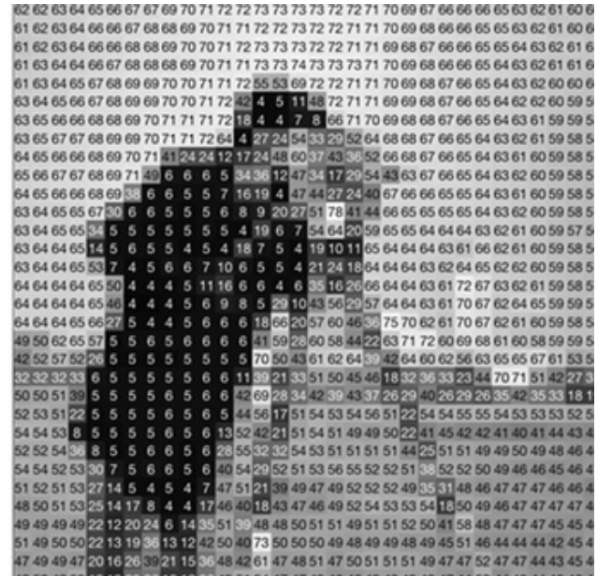


Figure 1.0: How computers view images [3]

Colour channels effect the value of pixels. In binary images, pixels are either 0 (black) or 1 (white). In greyscale images, pixel values range from 0 to 255, with 0 as black, 255 as white, and in between values representing different shades of grey. Colour images typically use the RGB model, with each pixel having distinct values for Red, Green, and Blue.

Being able to recognise the numbers is not enough for a computer to interpret the image but these pixel values can be used to extract a lot of meaningful information by using various computer vision techniques.

Image Recognition

Image recognition is the process of identifying patterns and locating objects in the numerical data that represents a digital image [5]. This achieved through the use of Artificial Neural Networks (ANNs), which are trained on a large dataset. The dataset consists of labelled images containing patterns and objects that we want the computer to recognise.

ANNs are a powerful Machine Learning (ML) model inspired by the structure of the human brain [5] [6]. These networks consist of many small software modules called nodes. These nodes are organised into layers, with each layer performing specific tasks. The first layer is the input layer that receives the pixel values from the input image. The output layer makes the final prediction. The in-between layers are called hidden layers, and they preform most of the computation required to make a prediction [6].

Nodes of one layer are connected to nodes of another layer through channels with an assigned value known as the weight [6] [7]. The input pixel values are multiplied with the corresponding weights and their sum is sent as input to the nodes in the first hidden layer. A threshold function is then used to determine whether a node is activated or not. An activated node is then passed to the next layer and this process is continues for the rest of the layers leading to the output prediction [7] [8]. This process is known as forward propagation [7].

In order for a network to make accurate predictions in most cases it requires training on a large dataset. Training involves feeding both input and output data to the ANN. The predicted

output is compared with the actual output, and errors are propagated backward through the network in a process called back propagation. Based on the error information, weights are adjusted until the network can make an accurate prediction in the majority of cases [8].

This process of forward and back propagation is repeated with multiple inputs until the network is trained on the large dataset, producing accurate predictions [9] [10]. One of the most popular ANNs is the Convolutional Neural Network (CNN).

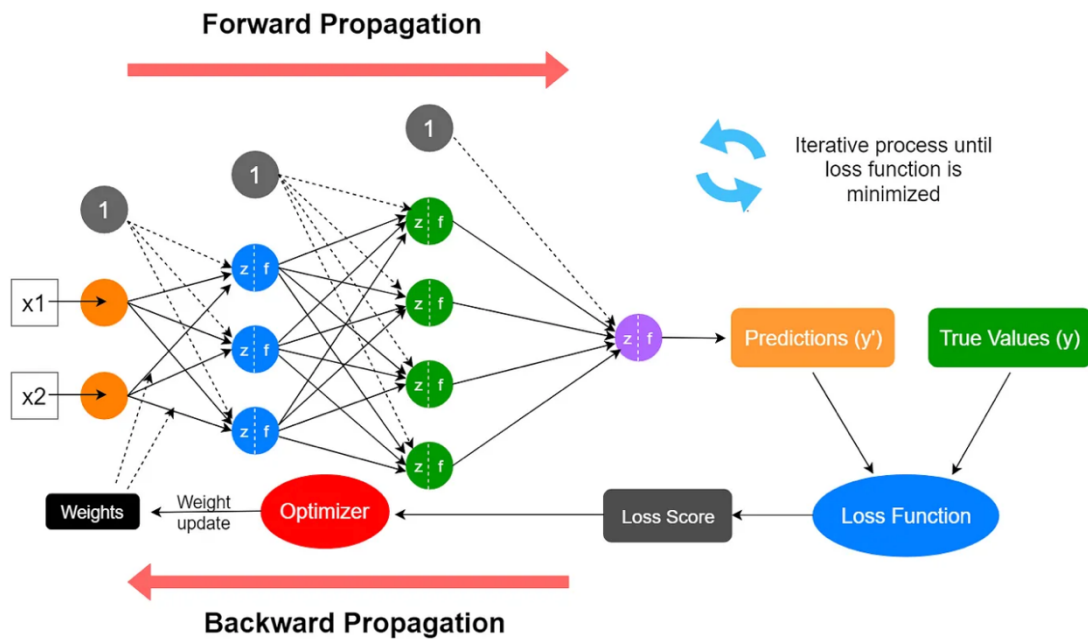


Figure 2.0: Process of an Artificial Neural Network [11]

Convolutional Neural Network (CNN)

One of the most powerful ANNs used for image recognition is the Convolutional Neural Network (CNN) [10] [12]. CNNs consist of a layered architecture enabling them to learn hierarchical features from data [10]. This means that simpler patterns such as lines, and curves are detected before more complex patterns such as objects, and faces. CNNs are constructed of three types of layers, convolution, pooling and fully connected [12] and it has superb performance in machine learning problems [12].

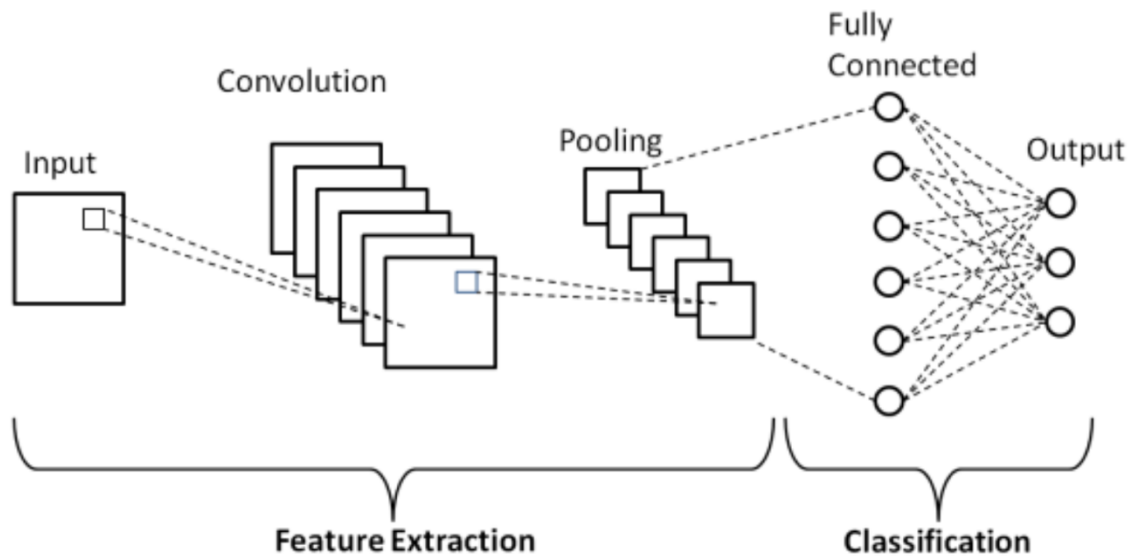


Figure 3.0: Convolutional Neural Network Diagram [13]

The core of a CNN is the convolutional layer. The convolutional layer uses filters called kernels, which are small grids of numbers, to slide over an input image and extract important features [10]. The values in the kernel are multiplied with corresponding pixel values in the input image. This process is applied across the entire image producing new sets of values. Different kernels can capture various features like textures, colours, or shapes.

CNNs usually have several convolutional layers, enabling them to learn complex features at different levels. The result of applying convolutional filters to the input image is a set of feature maps. Each feature map corresponds to the output of one filter applied to the input. The values in the feature map represent the presence or absence of a certain pattern/feature in the input. These layers are often followed by pooling layers, which reduce the size of feature maps while keeping essential information [10].

By stacking multiple convolutional and pooling layers, CNNs learn hierarchical representations of an image, capturing both basic and complex features for example edges and objects respectively [10]. Finally, these features are used in fully connected layers for classification or other tasks [10].

Image Processing

Image processing is a subfield of computer vision, that enhances and extracts meaningful information from an image through various algorithms. It is used to analyse and manipulate the contents of images or the image data and is an important step as the techniques involved boosts the performance of machine learning models [14].

The process typically starts with pre-processing steps such as resizing, noise reduction, colour correction and normalisation to ensure the data's quality and consistency. Next, feature extraction methods are used to identify distinctive patterns and characteristics that represent an area of interest [14].

The smart car parking system aimed to be developed accompanying this research will utilise image processing techniques to solve this paper's problem. Later in the literature view various image processing tools and methods will be explored in detail.

Image Segmentation

Image segmentation is the process of dividing images into meaningful regions based on characteristics such as colour, texture, or shape with a goal of simplifying the image for analysis. Image segmentation can be categorised into three types: instance, semantic and panoptic.

In instance segmentation, objects within an image are detected along their boundaries, this approach is valuable in application where individual objects need to be identified [15].

Semantic segmentation involves assigning a label to every pixel in an image, providing a comprehensive labelling of the entire image [15].

Panoptic segmentation combines elements of instance and semantic segmentation [16]. It assigns class labels to both objects and pixels and enabling the identification of individual object instances within the image. This approach is particularly useful in applications where computer vision models must recognise and interact with various objects in their environment.

Colour Segmentation

Digital images use the RGB colour model which uses a combination of distinct values for each red, green and, blue making the colour for each pixel. However, for colour segmentation it is common to convert the RGB image to the HSV (Hue, Saturation, Value) colour space. This is because HSV separates colour information from brightness making colour segmentation more accurate and robust in the case of lighting variation in an image [17] [18]. The output of colour segmentation is a binary mask with pixels within the range set to 1 and outside the range set to 0 indicating the presence or absence of the desired colour [18] [19].

Object Classification

Object classification is a subfield of image recognition that automatically assigns predefined labels or categories to objects in an image or video [20]. The process involves training a model such as a CNN on a labelled dataset, where each image has a label [21]. During training the model learns to identify and extract features from the input data that are characteristic of different object classes. Once the model is trained, it can be used to predict the class of new unseen images by analysing their features and comparing them to what it has learned during training [21].

Object Localisation

Object localisation is the process of identifying the location of a single object of interest within an image or video. This task aims to draw a bounding box around the object, indicating its precise spatial position [21]. The process is typically achieved through

regression-based techniques, where the model learns to predict the coordinates of the bounding boxes [22]. During training the model is provided with labelled data that includes the location of the object in each image, the model learns to associate the objects visual features with the corresponding bounding box coordinates [22]. Once trained the model can accurately localise the object in new images.

Object Detection

Object detection is the task of identifying and localising multiple objects within an image or video. It is a more complex task that not only assigns class labels to the detected objects but also draws bounding boxes around them to indicate their spatial locations [21] [23]. There are non-neural object detection models such as Histogram of Oriented Gradients and neural network algorithms such as YOLO (You Only Look Once) and SSD (Single Shot Multibox Detector) [21] [23]. These models process the input data at multiple scales and locations, searching for objects across the entire image. Once the object is detected, the model predicts its class and refines the bounding box to accurately localise it.

Certain non-neural methods may have limitations in accurately identifying objects or may generate a substantial number of false-positive detections. Neural network-based approaches while generally more precise, do have their drawbacks. Like requiring a large volume of annotated data for training, which can be both time and resource intensive [24].

Object detection can be divided into two main categories single-shot detectors and two-stage detectors. Single-shot detectors process an entire image in a single pass to make predictions about the presence and location of an object. With two-shot object detectors the image is processed twice, once for the location of objects and once to refine and make final predictions. Two-shot object detection produces a more accurate result but is more computationally expensive. For real-time applications single-shot object detection is better suited, while two-shot object detection is better for applications where accuracy is more important [25].

Image Processing Techniques

The proposed solution for the car parking problem is aimed to be developed using image processing and other computer vision techniques. This section of the literature review will look into various image processing techniques and tools in more detail in order to gain an extensive understanding to be able to implement the smart car parking system.

Kernel Convolution

Kernel convolution is a technique that is used a lot in the computer vision field. In image processing specifically it is used in techniques such as Gaussian blur and edge detection. Kernel convolution is a process that uses a small grid of numbers, i.e. a kernel and pass it over an image, transforming the image based on what the numbers are. By using different numbers in the kernel, we can perform blurs, and detect edges in an image. For the process of kernel convolution, the kernel is passed over every pixel in the image ensuring the pixel is centred. Because the pixel needs to be centred the kernel size is always odd [26]. For each pixel its neighbouring pixels are multiplied with the overlapping numbers in the kernel in pairs and these numbers are summed up. This value is then normalised by dividing it by the

total value of the kernel. After performing the calculation, the centre pixel's value is replaced with result of the calculation in a new image [26].

Blurring

Blurring is a crucial image processing step. Prior to applying techniques such as edge detection or thresholding, it is essential to smooth an image to reduce high-frequency content, such as noise and edges. By reducing the detail in an image objects of interest can be easily identified. While there are various smoothing and blurring options available, this paper will focus on three prominent techniques: Mean/Average Blur, Gaussian Blur and Median Blur [26].

Average blur

In average blurring the aim is to reduce details and noise in an image by relying on the pixel average. To achieve an average blur the image is convolved with a $N \times N$ normalised kernel. As the kernel moves across the image, the central pixel is set to the average of its surrounding pixels [27]. The size of the kernel determines the degree of the blur with larger kernels resulting in a more significant blur. In average blurring equal weight is given to all values in the kernel. Further discussions on varying weights are explored in the Gaussian blur [26] [27].

Gaussian Blur

Gaussian Blur, like average blur in image processing, is used to reduce noise and detail. Gaussian blur is more controlled, and edges are preserved more than in the average blur. Instead of a simple mean, it uses a weighted mean, giving more weight to pixels nearer to the central pixel [26][28]. So, the further away you get from the central pixel, the less weight is contributed to the combined average, hence why the edges are more preserved than in average blurring. The term 'Gaussian' is derived from the use of a Gaussian function. Like average blurring, Gaussian smoothing uses a kernel of $N \times N$, where N is an odd number. In constructing the kernel, a weighted pixel approach is used based on the distance from the central pixel. The Gaussian function used is as follows [28]:

$$G(x, y) = \frac{1}{2\pi\sigma} e^{-\frac{x^2 + y^2}{2\sigma^2}}$$

Here, x represents the distance from the central pixel in the horizontal axis and, y is the distance from the central pixel in the vertical axis and σ is the standard deviation of the Gaussian kernel [29].

Median Blur

Median blur uses an $N \times N$ filter, but unlike average and Gaussian blur it doesn't rely on kernel convolution calculation. Instead, each pixel's value is replaced with the median of the surrounding values within the filter's dimensions. This filtering technique is non-linear and

stands out for its effectiveness in noise reduction and excellent edge preservation making it a valuable tool in image processing [26].



Figure 4.0: Example of how powerful Median Blur is at reducing noise [30].

Edge Detection

Edge detection is a technique used to identify boundaries or edges within an image. Edge detection involves comparing each pixel's value to its surrounding values. An edge is detected where there is significant difference in values indicating transitions in intensity or colour [31]. These edges represent the areas where objects or features within an image are outlined or separated from their surroundings. A common operator for performing edge detection is the Sobel operator.

Sobel Operator

The Sobel operator detects edges in the x-direction and y-direction of an image by using a small kernel, commonly 3x3 making it a simple and quick process. The kernel for detecting edges in the x-direction uses negative values on the left-hand side (left column), positive values on the right-hand side (left column) and, zeros in the middle, while the kernel for the y-direction uses negative values on the top row, positive values on the bottom row and again zeros in the middle. This means that one side is slightly higher weighted than the other and the centre is preserved. The kernel is passed over every pixel to transform it using kernel convolution. If the result of the calculation is a large number, it indicates a significant change in intensity or colour [32].

-1	0	1
-2	0	2
-1	0	1

1	2	1
0	0	0
-1	-2	-1

Figure 5.0: Kernel for detecting edges along: x-direction (left) and y-direction (right) [33]

The above operation is performed in the x and y directions giving a result of edges in the x-direction and y-direction separately as shown in figure 6.0 below.

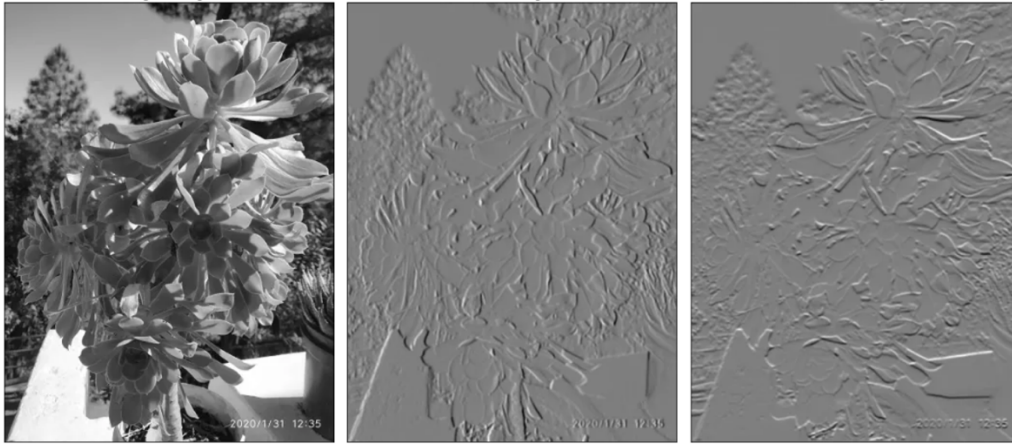


Figure 6.0: Edges detected separately around the x-direction and right direction [34]

The goal however is to have one image with both the edge directions combined so the total value. The magnitude of the gradient is calculated by [32]:

$$G = \sqrt{G_x^2 + G_y^2}$$

Where G_x is the value of the gradient in the x-direction and G_y is the value of the gradient in the y-direction giving the size of the gradient at that location.

By knowing these values, it is also possible to calculate the orientation of the edge by using the following formula [32]:

$$\theta = \tan^{-1} \frac{G_y}{G_x}$$

Where θ is the value of the angle and G_x and G_y are the gradient values along the x and y directions. The value of the angle says what the orientation in the image that particular pixel is which can be really useful for finding objects [32].

It is important to note that the Sobel operator is run on a greyscale image. This allows for the focus to only be on change in intensity. It is also common to apply a Gaussian blur prior to the Sobel operator for best result. This is because images can contain a lot of noise that might cause the Sobel operator to detect edges that are not really of interest. By using a blurred greyscale image, the final output will be a black and white image with white edges and dark everything else as shown in the figure 7.0 below.

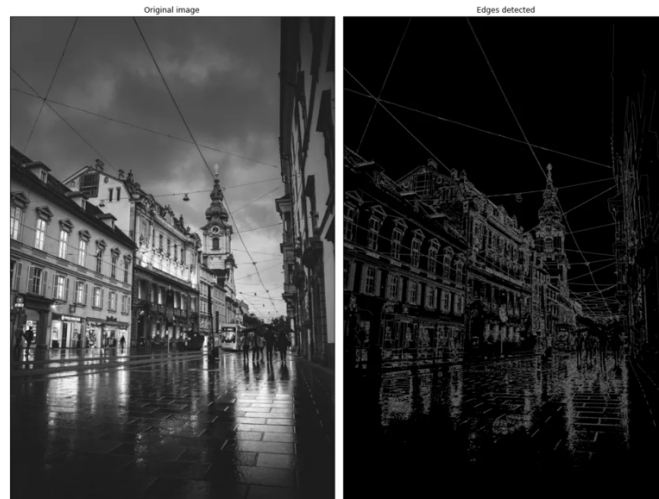


Figure 7.0: Binary image with edges in white and black everything else [34]

Canny Edge Detection

Canny edge detection is a commonly used edge detection algorithm created by John Canny in 1986 [35]. It is a multi-stage process that involves several steps [36]. The Canny edge detector improves upon the Sobel operator by refining the detected edges.

First Canny thins the edges so that they are one pixel wide, this is called non-maximum suppression [35][36][37]. This is done because what matters is the location of the edge and not its thickness. In this step, the gradient magnitude is used to thin down the edges. Edge points with a very small magnitude are suppressed to zero ensuring only the sharpest edges are kept.

Canny then uses a two-level thresholding process called hysteresis thresholding. Hysteresis thresholding is used to classify edge pixels into two categories strong edges and weak edges. For this process two thresholds are set, a high threshold and a low threshold. If the gradient magnitude is above the high threshold, it is classified as a strong edge pixel and is kept. If it is below the low threshold it is classified as a weak edge pixel and is disregarded. If the gradient magnitude is between the high and low thresholds it is only kept if it is connected to a strong edge pixel. This is because this indicates that these edges are most likely part of the same object. This process helps preserve only the most significant edges while eliminating weaker edges and noise. The output of the Canny edge detector is an image with thin dominant edges [35][36][37].

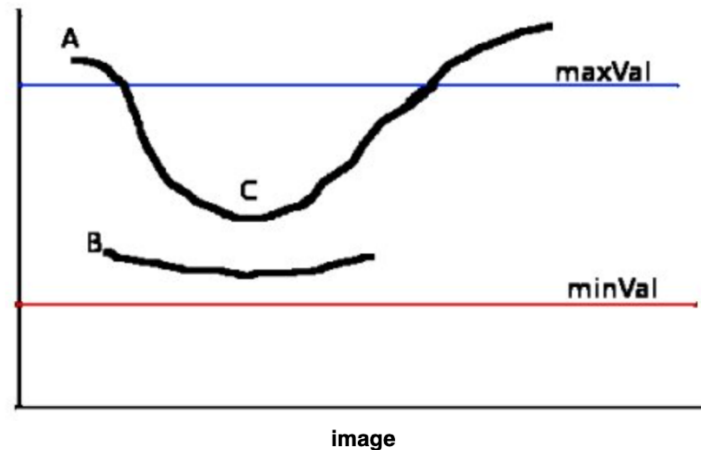


Figure 8.0: Diagram showing hysteresis thresholding A is a strong edge pixel, B is a weak edge pixel, C is in between but is connected to A therefore kept as valid edge [38].

Thresholding

Thresholding is the process of segmenting an image where pixels are set to either 0 (black) or 1 (white) i.e. converting an image to a binary form [26] [39]. Converting an image to binary form is called binarization and thresholding has a few algorithms that can perform this operation.

Binarization typically works on greyscale images, as coloured images have different values for the three colour channels. In a greyscale image each pixel has a value between 0 and 255. 0 being black, 255 white and anything in between is a different shade of grey. To convert a greyscale image to a binary image, any pixel value below 127 is set to 0 and any value above 127 is set to white. This process is called global thresholding and is the simplest form of thresholding. There are other simple thresholding algorithms for this paper the focus will be on adaptive thresholding.

Adaptive thresholding

The biggest drawback of simple thresholding methods is that we need to provide a threshold value. Manually providing a threshold value is not ideal, because it is not always possible to know what the best threshold value is. Moreover, images often contain lighting variations and setting one threshold possess limitation as the value of the threshold may work well for one part of the input image and not so well for another. Adaptive thresholding takes away this ambiguity. Adaptive thresholding uses a small set of neighbouring pixels at a time and calculates the threshold value specific to that area and then performs the segmentation. OpenCV's adaptive thresholding method takes in a few parameters, the input image, a maximum value, adaptive type, threshold type, block size, and a constant value (C) [40][41].

There are two common adaptive threshold types in OpenCV:

- `ADAPTIVE_THRESH_MEAN_C` where the threshold value is the mean of the neighbouring pixels minus the constant C.

- `ADAPTIVE_THRESH_GAUSSIAN_C` where the threshold value is a Gaussian weighted sum of the neighbouring values minus C .

Morphological Operations

Morphological operations are a set of mathematical techniques designed to manipulate the shape of features in binary images. In morphological operations a structuring element (a kernel) is applied to an input image to produce an output image of the same size. The value of each pixel in the output image is determined by comparing the corresponding pixels in the input image with its neighbours. The purpose of these operations is to enhance a shape to be able to segment it from the background. There are two types of morphological operations: Dilation and Erosion [42].

Dilation

In morphological dilation the shape of an object in an image is expanded. The dilation process requires two inputs: the image itself and a kernel, often referred to as a structuring element, which dictates the impact of dilation on the input image. Commonly the input image is a binary image, but it can also be performed on greyscale images. Binary images consist of pixel values 0 (black) and 1 (white), representing background and foreground respectively. The expansion is achieved by comparing each pixel in the input image with corresponding pixels in the kernel. Typically, a 3x3 matrix serves as the kernel in dilation, though larger kernels can be used for a more intense expansion. As the kernel traverses the input image, if any of its pixels coincides with a foreground pixel the surrounding background pixels are changed to 1, thereby expanding the shape of the object [42].

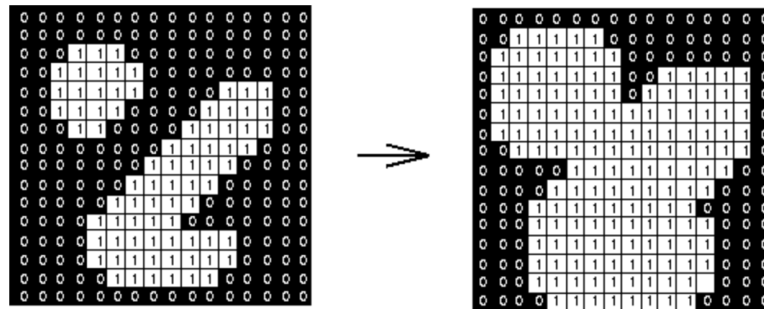


Figure 9.1: Morphological dilation process of expanding the shape of objects [43]

Erosion

In contrast, erosion is the process of shrinking the shape of the object in a binary image. Similarly, to dilation a comparison is made between the input image pixels and the kernel. The output is set to 1 only if all the corresponding pixels in the structuring element and the input image are set to 1 [42].

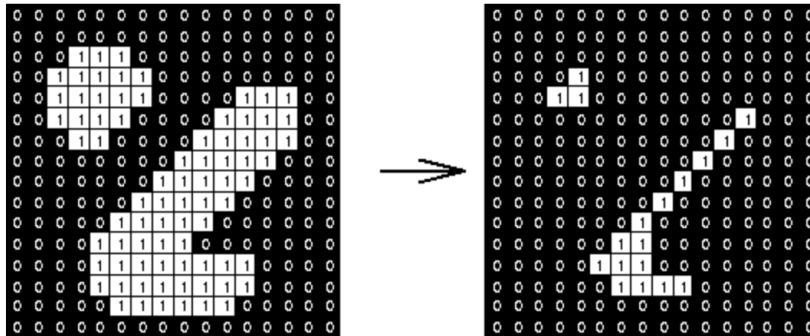


Figure 9.2: Morphological erosion process of shrinking the shape of an object [44]

Implementation Frameworks and Libraries

While background theory is important to understand the methods that can solve the problem of this paper, it is equally as important to explore the practical frameworks that can be used to implement a smart car parking system. This literature review section will examine frameworks that can be used to implement this system. Specifically, we will look at the Python language and the libraries it supports for the backend, look at the Flask framework for RESTful web services and explore OpenCV for image processing.

The main goal of a smart car parking system is to inform and guide users to available parking spaces speeding the time spent locating an available space. The aim is to develop a RESTful web service to update users of available parking spaces in real-time this involves a backend and a frontend.

Backend

The backend of an application, also known as server-side, refers to the part of an application that is hidden from the end-user. The backend handles processes that ensure an application functions as intended. Therefore, choosing the right programming languages, frameworks, and libraries for the backend is crucial for building a robust and efficient application.

The most popular language for Artificial Intelligence (AI) and Computer Vision is Python due to its simplicity and versatility. Python supports many libraries used in image processing and computer vision [45]. Some of the most popular libraries include OpenCV, Numpy, and SciPy [45]. These libraries provide built-in functions and modules for machine learning, image processing and other tasks. Python can be used to build web, mobile and desktop applications. For this reason, Python is the language of choice for building the smart car parking system application.

Flask Framework

Flask is a lightweight Python web framework that is commonly used to build web applications and APIs. Flask does not come with a lot of built-in features giving developers the flexibility to integrate the libraries and components needed. It is well-suited for building RESTful APIs as it makes it relatively easy to define routes and handle HTTP requests. This will be particularly useful in the case of the smart car parking system to send parking space updates to the fronted of the application in real-time.

OpenCV

When it comes to image processing and object detection for the application, a popular library that can be integrated is Open-Source Computer Vision Library or OpenCV. OpenCV is a popular open-source computer vision library for image and video analysis originally developed by Intel in 1999 [45][46]. It is a cross-platform library that can be run on Windows, IOS, Android, Linux and other operating systems and has over 2500 [47] algorithms that are utilised in computer vision and machine learning. These algorithms can be used to process images, identify objects, recognise faces and much more.

There are many image processing techniques that can be achieved in OpenCV, for this project OpenCV will be utilised for processing the data required for building a smart car parking system. Considering the timeframe for this project, it makes sense to utilise OpenCV for the development stage. OpenCV's built in algorithms for blurring, edge detection and other image processing techniques will assist in building a working prototype application.

YOLO

You Only Look Once (YOLO) is a popular CNN-based real-time object detection model. YOLO is a single-shot detector meaning it processes the entire image in one forward propagation to detect objects [48][49][50]. This makes it much faster at detecting objects however the results of the detection are less accurate than that of a two-stage detector.

YOLO divides the image into many regions using bounding boxes and makes a prediction for each region. To achieve this the image (or video) is divided into a grid of $S \times S$ equal cells. Each grid cell is responsible for predicting bounding boxes all containing an object as well as a score representing the level of confidence for the prediction. If there are no object in the cell the score should be zero. For the cells that do contain an object, YOLO calculates the Intersection Over Union (IOU) for each grid cell. IOU is the intersection area of the predicted box and the ground truth divided by the union of the two as shown in figure 10.0 [48][49] [50].

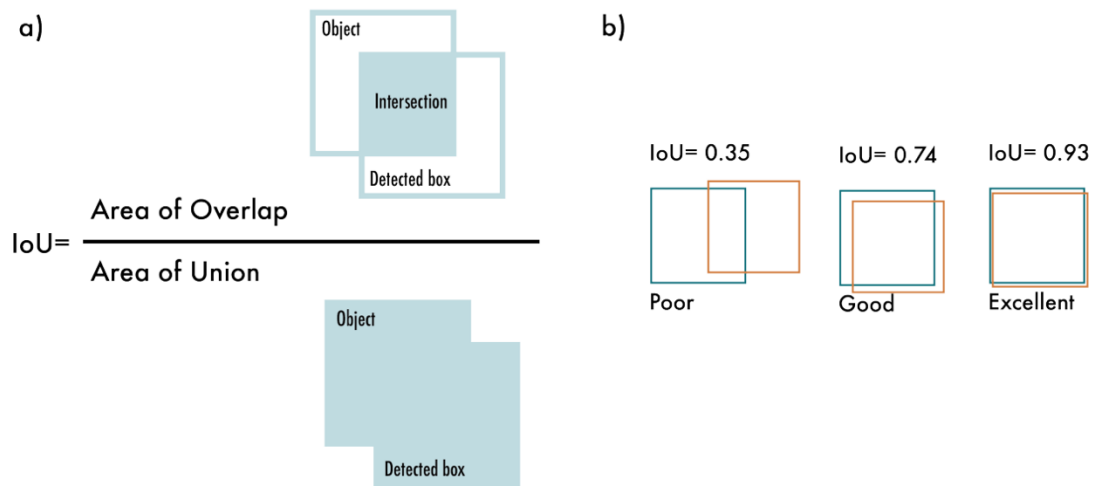


Figure 10.0: Diagram illustrating intersection over union (IoU) formula [49]

To filter out irrelevant boxes and keep only those that are meaningful a threshold value is set. Grid cells with an IOU values less than or equal to the threshold are ignored while values greater than the threshold are considered relevant. When multiple IOU values are greater than the threshold, non-maximum suppression is used to keep only the boxes with the highest probability score of detection. The output image (or video) will be the input image with objects localised and the classes attached to each bounding box [49].

Though the accuracy of YOLO is slightly lower than other two-shot detectors such as R-CNN and Fast-RCNN, YOLO is much faster at detecting objects than these methods making it a suitable algorithm for detecting cars in the case of the smart parking system of this project [48][49][50].

Frontend

The frontend of an application, also known as client-side refers to the parts of the web application that is visible and intended for the end-user. A frontend framework consists of pre-written code and tools helping developers with creating user interfaces of a web application swiftly. There are many popular frontend frameworks such as React Native, Angular, Next.js and many more.

Given the time constraint for this project, the prototype application will focus on building a robust backend for real-time parking space status updates. For this reason, the frontend will be on the simpler side. The jQuery library coupled with AJAX for interacting with the backend alongside HTML and CSS for creating a friendly user interface will be used for the prototype of this application.

5. Existing Solutions

A smart parking system is a system that uses modern technology to effectively speed up the time looking for an available parking space, saving time, obstruction, and fuel consumption. The technologies involved may include cameras, infrared, sensors, monitors and mobile applications [19]. Smart parking systems are excellent because of their ability to update users with available spaces in real-time. There are different types of smart parking systems, [19]. conducted a study of three categories, counter-based, sensor-based, and vision-based smart parking systems which are believed to be most suitable for our problem.

Counter-based Systems.

A simpler system for managing and monitoring parking space availability in a car park is a counter-based car parking system. This method involves counting the number of cars that enter and leave the parking area. Sensors are strategically placed at the entrance and exit barriers to track the movement of vehicles. While these systems can display the number of available spaces, they do have a limitation of not providing guidance to drivers in finding an available spot [2].

Sensor-based Systems

Sensor-based systems are systems where sensors are placed on each parking space to detect whether a car is present, and the availability is displayed in various ways to clearly direct drivers to the spot [2]. [52] proposed a IoT based car parking system using infrared (IR) sensors. It includes a microcontroller (nodeMCU), an IR sensor, and an LCD screen. The IR sensor detects if a car is present or absent in a parking spot, and the LCD screen displays which slots are available. The system continuously monitors the parking spaces and updates the information on the LCD screen. This system assists drivers to find a spot quickly and keeps a record of the parking spot availability.

Although sensor-based systems have an advantage over counter-based systems, as it can guide drivers to the parking spot directly, they can be extremely costly as it is required to install sensors and screens on each parking spot to be able to monitor it.

Vision-based systems

In vision-based computer vision techniques are developed to determine the availability of spaces and to guide drivers to the right spot. This type of system uses camera and image processing algorithms to analyse the space. Cameras are strategically placed in the parking area to capture images or videos of vehicles and surrounding areas. The captured footage is processed using computer vision algorithms. These algorithms analyse the images and identify key features such as vehicles, obstacles, and spaces. The system analyses the size, shape, and position of the parking space and its availability and detects vehicles in the parking area as well as its movement and position in real-time. Vision-based systems therefore require cameras, image processing techniques, advanced machine learning algorithms and a display or user interface.

Though vision-based systems maybe expensive to implement due to the large amount of data that will be transmitted over a wireless network, most car parks already have CCTV cameras installed which lowers installation costs [2]. Vision-based systems can be further divided into two categories, vehicle detection and space detection where the former uses computer vision to detect if a vehicle is present in a spot and the later uses colour detection to determine if a space is occupied or free [2].

[53] developed a vision-based carparking system to detect available parking bays and their location. They used feature extraction and object identification techniques to achieve this [53]. For feature extraction they used a special pattern called Haar-features to detect and identify cars in parking spaces. Haar-like features helps classify the information in an image. They are used to determine the presence of a car by analysing regions of the image. A group of Haar-like features is used to find these characteristics, and the resulting information is fed into a decision tree classifier to identify cars. Open Computer Vision Library (OpenCV) was used to help with object classification. To train the system, they needed two sets of images positive and negative. Positive images show the object they want to detect and negative images that do not show the object. During training, they also needed to specify where the object is located in the positive images and provide its height and width. Their results showed the high level of accuracy in determining the presence of a car however due to the use of web cameras to monitor car parks, this showed some weaknesses in terms of accuracy of the detection due to the limitations of the camera.

Vision-based systems are cost-effective, accurate, and highly scalable giving them an advantage over counter-based and sensor-based systems.

6. Methodology

Finding a parking space, especially in a large car park, can be a frustrating process particularly at the beginning of a working day. This universal problem is also applicable to the DVLA car parks where one can sometimes spend more than 15 minutes looking for a parking space. The DVLA parking problem is what sparked the idea for this research project. The aim of the research is to find a solution for this problem by designing and implementing a smart car parking system to inform users of available parking spaces in real-time saving them time and frustration.

In order to design and implement a practical and robust smart car parking system this research project will be divided into three stages, literature review, software development and result analysis.

Literature Review

The literature review is the base of this research. Its purpose is to gain a deep understanding of both the background theory and practical solutions relating to the smart car parking system. Therefore, a detailed research of previously published papers on Computer Vision, Image Processing and Machine Learning has been conducted. Additionally, it is imperative to

conduct a study of existing solutions to be able to find an optimum solution. Finally in the literature review stage web frameworks and programming languages have been researched to find the best tools for the implementation stage.

A comprehensive grasp of the topic during the literature review will serve as the foundation for the subsequent stages.

Software Development

The primary objective of this research is to find a solution for locating available parking spaces, so it makes sense that an application is created alongside the research making the implementation a pivotal step. A proof of concept will be developed to serve as a prototype for the application. This prototype will make use of various image processing and object detection techniques to identify and mark parking bays as either occupied or available and update users in real-time.

Initially the plan was to come up with a solution using colour segmentation techniques however following a conversation held with the Emerging Tech Lab Team at the DVLA it was mentioned object detection for a smart car park system is the preferred direction. Since the DVLA office car parks is what inspired the idea for this project, this shifted the plan to create two systems one with object detection and one with colour detection to compare results and determine the best solution.

Result Analysis

In the final stage of this project, the two systems that are created will be evaluated and tested to measure the accuracy of detecting a parking space and determining its status. By gathering the results, it will guide to the future improvements and additions required for the next iterations to work towards a finished product.

As two systems will be created the results will be collected as follows.

- The accuracy of detecting a parking bay in an image or video at a bird's eye view (0° angle), a bird's eye view with a slight angle (approx. 10-20° angles) and at a 45° angle
- The accuracy of detecting a car in a parking bay using a pre-trained object detection model
- The accuracy of the bay status detection by using colour detection in the ROI and comparing with the ground colour
- The accuracy of bay status updates in various lighting conditions

7. Design

For the design of the prototype, it was decided to take a feature driven development approach as this will help with building on the application iteratively. Each feature will be divided into smaller tasks to make the development process more manageable. Figure 11.0 below illustrates the ideal process for implementing the two systems.

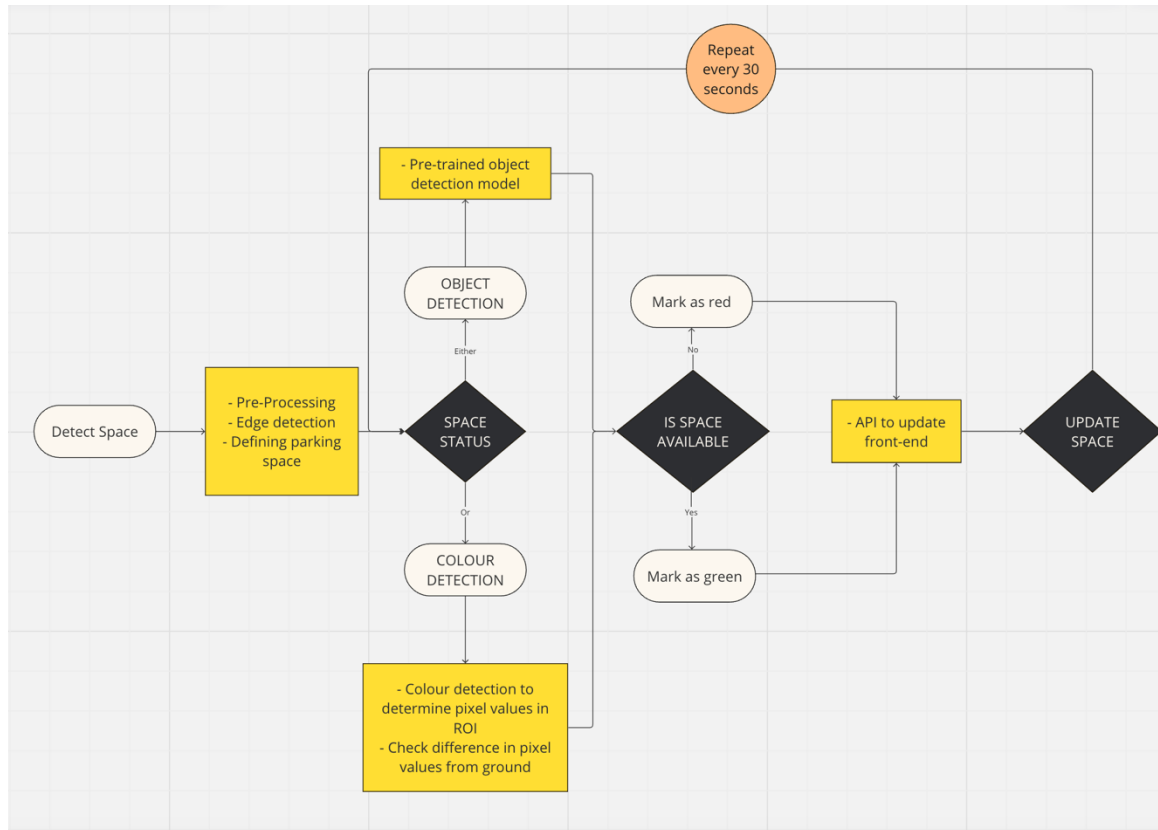


Figure 11.0: Prototype design for the smart car parking system.

Features

Feature 1: Space/Bay Detection

The region of interest (ROI) for the application is a parking bay. Therefore, the initial step is to determine what qualifies as a parking space and what doesn't. Parking bays are usually separated with lines or full rectangles, these lines will be used to define the ROI.

For detecting a parking space there are a number of tasks that need to be completed:

- Image processing
- Edge Detection
- Image Segmentation
- Object Localisation

Since the image of the car park contains many other elements than the parking bays, Canny edge detection will be used post pre-processing steps such as reseizing, greyscale conversion, and Gaussian blur to reduce the elements to be processed. Next will be to find the contours of the detected edges as the contours will help with locating shapes. Once all contours are found they can be filtered to find contours of a specific size to match that of a parking bay. Once the size has been determined and all contours of defining the parking bays are found, bounding boxes will be drawn around each bay giving its location coordinates.

Feature 2: Occupancy Status

Feature 2.1: Colour Detection

The background colour for the chosen car park is grey, by using the HSV (Hue, Saturation, Value) of grey, first high and low thresholds for it will be saved. These values will allow to search for any values falling within this range. Secondly, each bay will need to be cropped along its coordinates that were located in feature one. Each cropped bay will also be converted to HSV, and its values compared to the high and low thresholds set. If the bay pixels fall within the range the space can be marked as available else unavailable.

Another option for colour detection is by applying an adaptive threshold and median blur to the car park image to segment the objects from the background. By then applying dilation to the threshold image, the size of objects that are detected can be expanded. Then using the coordinates of each bay found in feature one can be used to calculate the non-zero-pixel values in each bay and by defining a threshold if the values fall below the threshold the bay can be marked as available else unavailable.

Feature 2.2: Object Detection

This feature was the preferred direction for the team at the DVLA. Object detection is a complex task that may be beyond the scope of the prototype. However, by using a pre-trained object detection model this feature may be achievable. Initially the YOLO model was

favourable for this step however using OpenCV's Cascade Classifiers may be a more realistic solution given the time constraint for this project. This technique uses pre-trained classifiers to detect cars and draws bounding boxes around them [54].

Feature 3: Marking Bay as Occupied or Unoccupied

For the options of feature two it is possible extract relevant data to determine the status of each parking bay.

For the dilated image, a threshold can be set to determine the space's availability. Spaces falling below the threshold can be marked as available whilst those above the threshold means the space is occupied and so green and red bounding boxes can be drawn around the spaces respectively. For the HSV model pixel values falling outside the range will be marked as 0. This can be used to mark those bays as unavailable and available otherwise.

For the object detection model, once the pre-trained classifier detects a car in the cropped image of a space, that space can be classified as occupied and if no car detected it will be available.

Once the space status has been determined whether by colour recognition or object detection, rectangles will be drawn around each space to highlight its status. Red rectangles will represent occupied and green rectangles will represent available.

Feature 4: RESTful Webservice

After implementing the features for marking bays and checking their status, the subsequent task involves creating a system to regularly update users on the car park's space availability. While developing a mobile application would be an optimal solution, due to time constraints a simplified approach is sufficient for the prototype. Therefore, a simple API will be developed to fetch space updates from the backend and display each space's status on a web page. To achieve this, FLASK will be used as it is a lightweight web framework. The frontend will be a straightforward HTML page, and, with the use of AJAX, GET requests will be made to the endpoint retrieving space updates. The aim is to make these calls every 30 seconds to mimic real-time updates.

8. Data Analysis

The initial step crucial for the successful implementation of the smart car parking system is data. This data consists of images and videos capturing a car park with different scenarios. It also requires the camera angle to be strategically positioned at an optimal height and angle to ensure a comprehensive view of each space.

While high-definition media is ideal, practical constraints in the real-world such as new equipment and installation costs may dictate a more cost-effective approach. Hence by using

already available footage such as CCTV footage coupled with image processing techniques proves to be a better solution.

During the project's initial phase, a meeting was held with the Emerging Tech Lab team at the DVLA to discuss the idea and explore potential data sources. Because the idea was sparked by the DVLA car parks it would have been nice to use images and videos of one of their car parks. However, due to security measures, using actual DVLA footage or taking pictures of the car parks at the DVLA is not feasible. Although some public images of the DVLA car parks are available online, the images only cover one of the car parks at a 45° angle this limited data poses challenges.

As a workaround, an external video has been found online that will be used. This video contains lots of ideal scenarios such as empty parking spaces, occupied spaces, cars parking up, cars leaving and individuals walking on parking bays. This will make sure that the prototype application has many scenarios to test and gather results from.

Although this video contains many scenarios there are some drawbacks, mainly there is not a single frame where the parking bays are all unoccupied which would be ideal for detecting the parking bays. Additionally, the video has a slight angle which makes some of the cars parked overlap over the bay lines of neighbouring bays and so covering edges as highlighted in yellow in the figure 12.0 below. In some frames there are individuals stepping out of their car as highlighted in orange figure 12.0 and are stood in the neighbouring bay which might cause a false update of that space when using threshold to determine the space status.

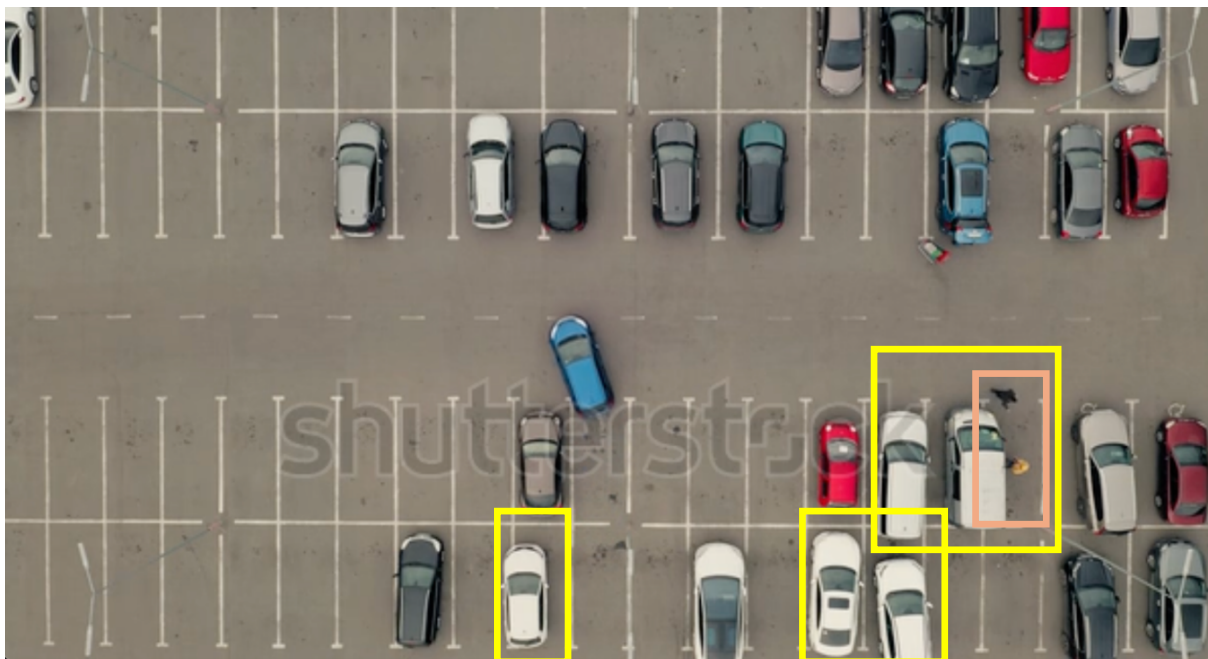


Figure 12.0: Car park image to be used for application highlighting potential drawbacks.

9. Implementation

Bay Segmentation/Detection

The ROI for this application is the parking bays, to detect these in the data the work was divided into multiple steps using OpenCV.

The obtained still image of the car park is a screenshot so it is important to resize the image. This initial step is crucial as once each bay is detected its coordinates in the image will be saved, if the image is not the same size as the video, the bay coordinates will not match the ones in the video later. This step was made easy by using OpenCV's resize function passing it the image and the required size.

Then it was time to convert the image to greyscale using OpenCV's colour conversion function and blurring the image using OpenCV's Gaussian blur with a 3x3 kernel. These processing steps are important to reduce noise so that the Canny edge detector only detects meaningful edges.

The next step was to detect the edges in the image. This was done using the Canny edge detector in OpenCV. It required the greyscale, blurred image along with a minimum and maximum threshold for the detecting the significant edges. For this particular image, 150 and 255 were used for the minimum and maximum thresholds. This resulted in the detection of meaningful edges as shown in figure 13.0.

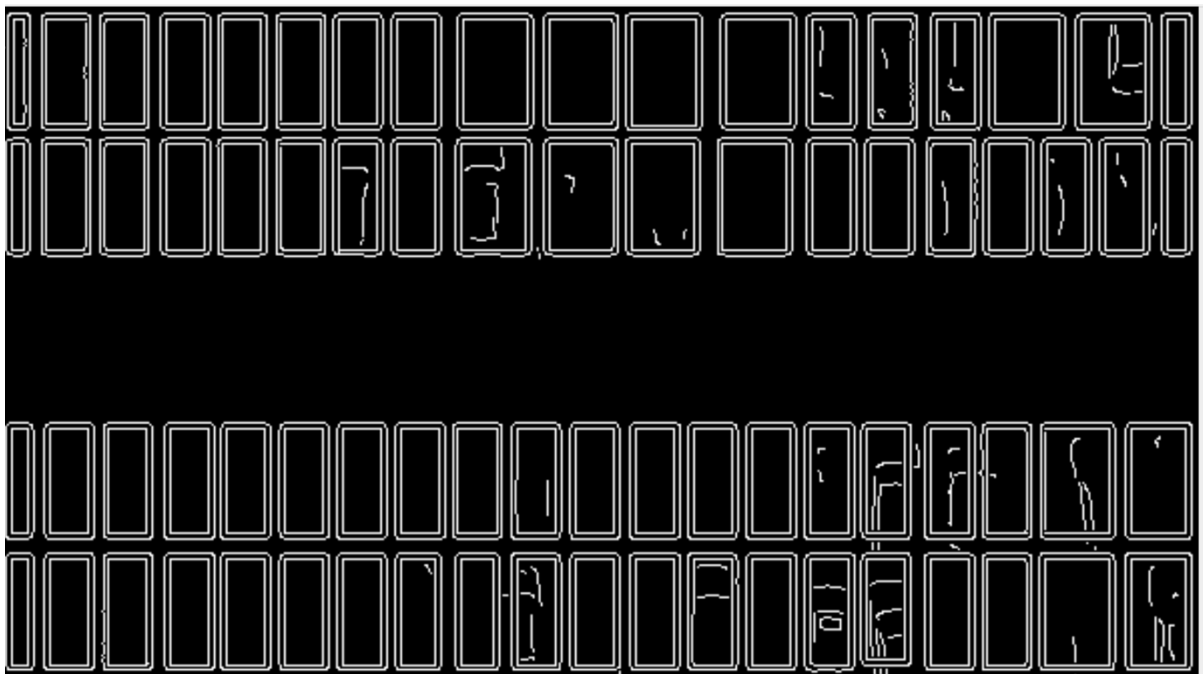


Figure 13.0: Detected edges using Canny edge detector.

Because the input image is not of a completely empty car park, Canny also detected some of the parked car edges within some of the bays. To remove the unnecessary edges I was able to use OpenCV's `findContours()`. This function looks for the contours in the binary image above. Contours are the curves joining continuous points with the same colour or intensity [55]. The output of this function will contain a list of all the contours with each individual contour consisting of a Numpy array of it's x and y coordinates. This function also takes in other parameters `cv2.RETR_EXTERNAL` and `cv.CHAIN_APPROX_SIMPLE`. `Cv2.CHAIN_APPROX_SIMPLE` makes sure only the important contours of specific line are stored getting rid of redundant points. `findContours()` uses a hierarchy system of detected contours, giving a relationship between contours detected within contours. Looking at the one bay in figure below we can see the outer edge, and some edges within it, all these will have contours. We call the car edge contours the children of the outer edge contours (the parent). By using `cv2.RETR_EXTERNAL` flag only the outer, or parent contours will be kept, which is exactly what I was looking for [56].



Figure 14.0: Image illustrating outer edges of bay and some car edges within it.

Now that I only have the parent contours, I still encountered the issue of the edges with contours that were detected outside of a parking bay some of which are highlighted in red in figure 15.0 below.



Figure 15.0: Some edges outside the bays with significant size difference.

There is a significant size difference between these contours and that of a parking bay, so by finding out the size of the contour I was able to filter through them and only keep the desired contours. The contour area is calculated by `cv2.contourArea()` method [57]. By finding out the contour sizes I was able to set a threshold to keep only the desired contours.

With the desired contours I was now able to draw a bounding rectangle around each bay and was able to extract its (x, y) coordinates along with its width and height. I then began contemplating how to save each bay's coordinates, so that they can be used for the next feature. I then discovered Pickle, a Python library used for serialising and de-serialising a

Python object structure. It is used for converting a Python object for example, lists, into a character stream. This can then be used in another Python script and converted to its original form [58].

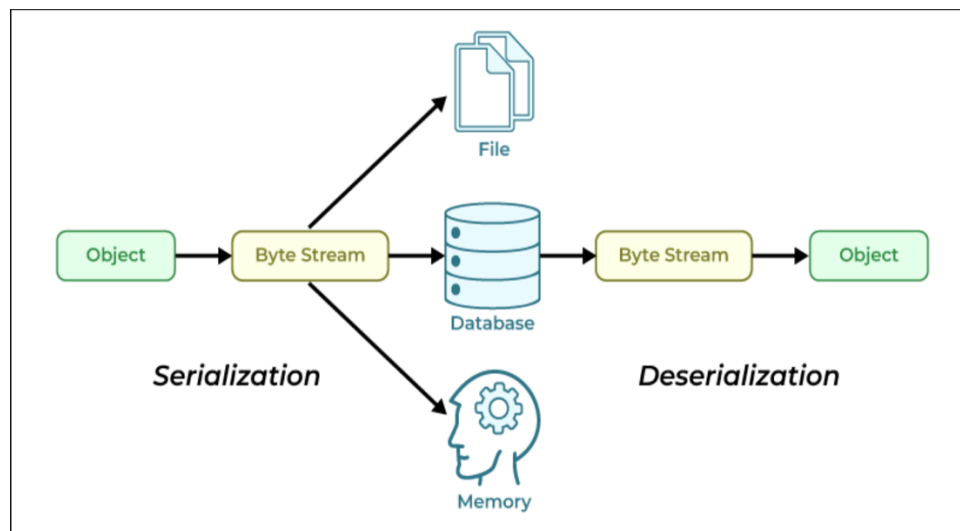


Figure 16.0: Converting a Python object (list in this case) into a character stream in Pickle [59].

I saved each contours coordinates into a Python list this list was then ‘dumped’ into a pickle file for use in the next feature development.

Determining Space Status

For determining the occupancy status two segmentation methods were used. The first approach involved video processing through the application of median blur, thresholding, and dilation to exaggerate the object shapes. A threshold was set for determining the space occupancy by calculating the non-zero-pixel values in each bay. Values below the threshold meant the space is available and above the threshold indicate the space is occupied.

The second method used the HSV colour channel to determine an upper and lower value for the ground colour of the car park. Any pixel values falling outside the range determined the space as occupied and values within the range determined the space as available.

Method 1: Calculating non-zero pixels in ROI.

Now that I had all the parking bay coordinates saved, it was time to work on the video footage. A video is merely a collection of still frames so it also required some image processing steps. By applying thresholding, blurring, and dilation. Once the video was loaded using OpenCV first it was converted to grayscale and the Gaussian blur was applied.

To identify occupied bays, a pixel calculation for each bay was necessary . To do this I needed the frames to be binarized, so I used the OpenCV’s Adaptive threshold method with the threshold type ADAPTIVE_THRESH_GAUSSIAN_C. Then a median blur was applied

and finally the image was dilated to emphasize the detected objects. For dilation a 5x5 kernel of 1's was used to really enhance the shapes of cars that are parked in certain bays.

The next step involved determining the status of each bay and marking it based on availability. For this I created a function that takes as inputs: the processed frames and the coordinates for each bay. Pickle was used here again to load the file with each bay's coordinates. In this function each bay was cropped along its coordinates and for each cropped bay the number of non-zero pixels in it were counted using OpenCV's `cv2.countNonZero()` function. By counting the pixels, it was possible to set a threshold for determining whether that particular space is occupied or available. Because there is a slight light variation in the original video a different threshold had to be set for the top and bottom bays in the video. For the threshold set, pixel values above the threshold declare the space as occupied and values below the threshold declare the space as available and by using OpenCV's `cv2.rectangle()` function red or green bounding rectangles were drawn around the bays to mark them as occupied or available respectively.

Method 2: HSV range for ground colour

The occupancy detection process using the HSV model involved minimal processing. Initially, the video frames were converted to the HSV colour space using OpenCV's colour conversion. Given that the ground colour of the car park in the video is grey, I was able to establish high and low HSV colour ranges for it.

Similar to the previous process, each bay was cropped, but this time each bay was also converted to HSV. Using the OpenCV `inRange()` function, I supplied it with the new HSV values for each bay, along with the defined low and high HSV values for the grey colour. The result was a mask indicated the presence or absence of the grey colour range. If no other colour was detected, the value was set to 1 otherwise it was set to 0.

Since the video was captured from a bird's eye view, some false detection occurred due to the residual grey areas in each bay. To achieve accurate detection, I used Numpy, a library for array manipulation. Using `numpy.all()` and providing it with the mask and the conditions [60], I successfully achieved mostly accurate detection. This proved to work however it worked the other way round all values equal to 0 were of bays without cars, and values over 0 were the ones with cars. This is something I have not been able to figure out as of yet.

RESTful Service

The prototype application was implemented with the focus mostly on the backend. The Flask framework was used to be able to create a simple frontend and an API to communicate with the backend of the application. The frontend simply consists of a HTML page with some CSS styling.

Asynchronous JavaScript and XML (AJAX) was used to make a GET request to an endpoint defined in the backend to retrieve a JSON containing the number and status of each parking space. The retrieved object is then split and converted to text and displayed nicely in a HTML

table. The call to the endpoint is made every 30 seconds to mimic real-time updates of parking spaces for the end users.

On the backend of the application endpoints were defined for updating and populating the table using the methods described above to make the changes.

10. Results and Discussions

For testing the prototype application to see how accurate it is at detecting spaces and checking their status multiple images of car parks at different angles were used. The same image pre-processing technique of manually highlighting each bay was used for all images to find out if this technique made a difference in the detection of bays. A total of 10 new images were used at 0°, approx. 20°, and 45° angles with large and small car parks for both the colour segmentation by non-zero-pixel values and the HSV method.

Table 1.0 below demonstrates the gathered results for the bay and status detection by counting the non-zero pixels in the ROI. The results determined a higher success rate for images taken from a top view. It also shows that the space detection method is more successful than determining the status of each space.

<i>Total Spaces</i>	<i>Total Vehicles present</i>	<i>Image Angle</i>	<i>Space detection accuracy (%)</i>	<i>Status Detection (%)</i>	<i>Error Space Detection (%)</i>	<i>Error Status Detection (%)</i>
20	0	Birdseye (0°)	100	50	0	50
14	7	Birdseye (0°)	100	92.8	0	7.2
50	21	20° (Approx.)	92	66	8	34
44	1	20° (Approx.)	95	52.2	5	47.8
10	1	45° (Approx.)	100	50	0	50
40	9	45° (Approx.)	85	47.5	15	52

Table 1.0: Results of space and occupancy detection of car park images at various angles

Figure 17.0 illustrates the results for the function used for detecting the parking spaces in all images. The average percentage of bay detection exceeded the target I had set. The target was 90% and the average result came to 95.3%. This mean the methods used were successful.

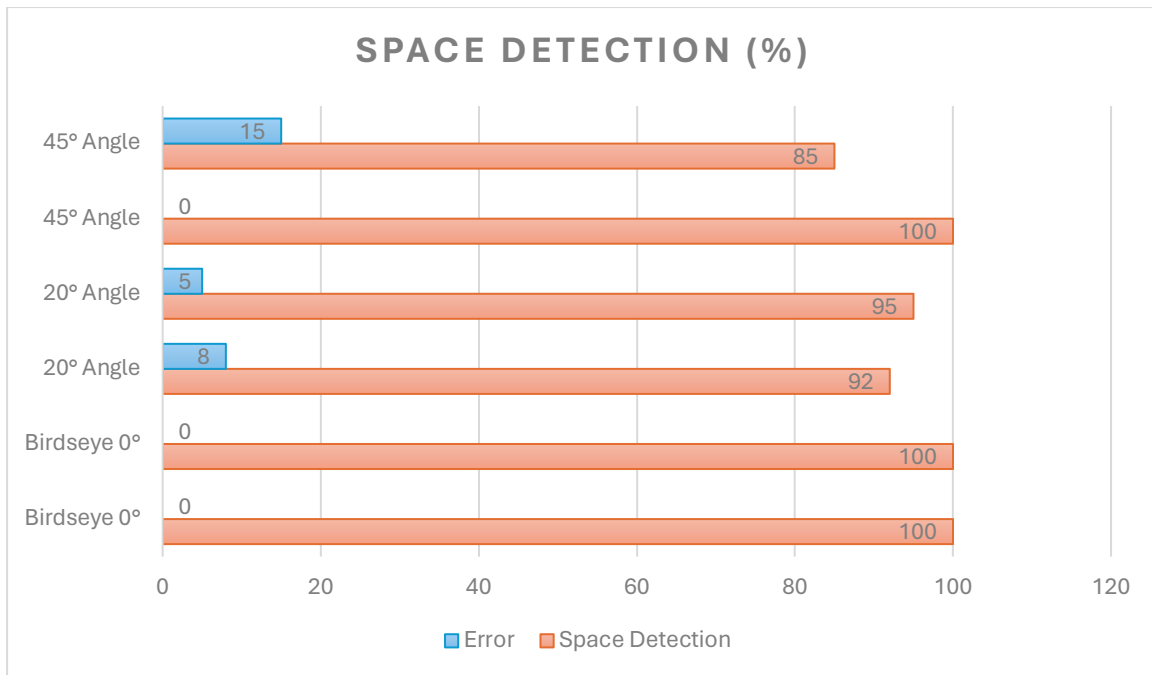


Figure 17.0: Results of detecting spaces in new car park images at various angles

The results for testing the accuracy of bay status detection were not as satisfactory. The target was 90% accuracy, but the actual turned out 59.75%. I think for this application the parameters and thresholds set for determining the status of a parking space would need to be adjusted for each car park hence the results. Figure 18.0 below illustrates these results. The accurately detected statuses were on average just over half of the spaces.

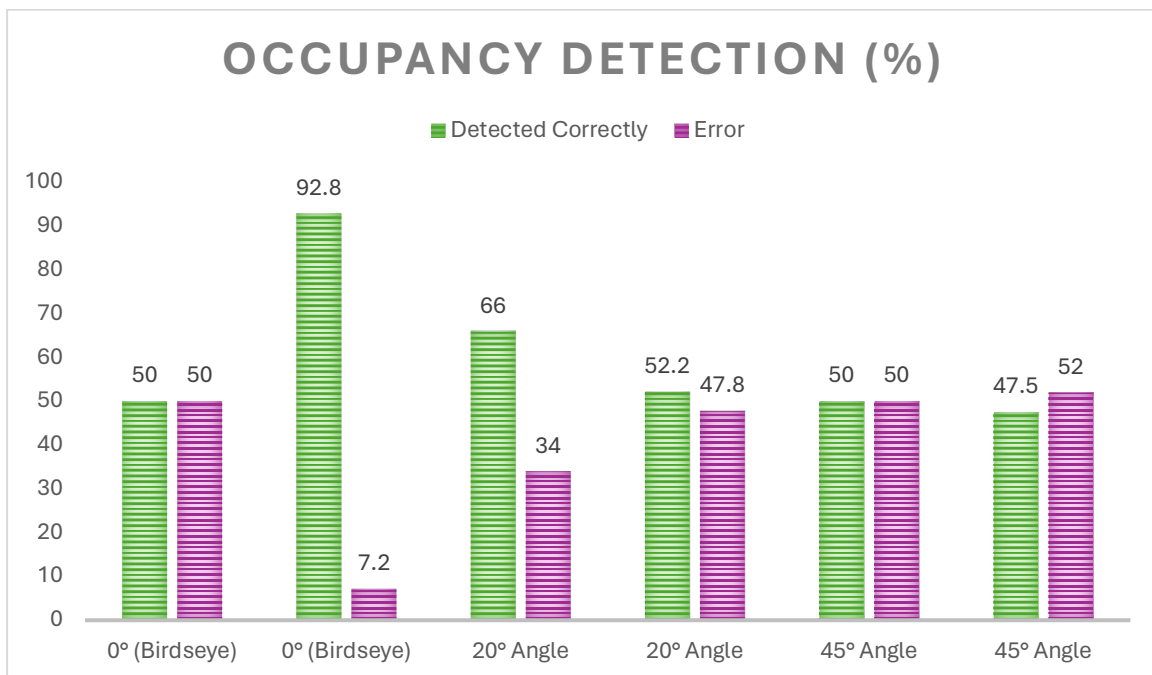


Figure 18.0: Results of occupancy detection in new car park images at various angles

The errors for both the space and status detection results were calculated by using the following formula:

$$EP = \frac{T - C}{T} * 100$$

Where EP is the error percentage, T is total spaces and C the spaces detected correctly. This resulted in 4.7% EP in space detection and 40.1% EP in occupancy detection.

When it came to using the HSV model for detecting colour within a specific range, occupancy detection performed a lot better than with calculating non-zero values for still images. However, in the video of the original car park, spaces with individuals walking on and cars slightly over lapping whilst driving passed a bay also caused a number of errors. Test images with cars parked within the range also caused the occupancy status to be wrong in some cases.

To test the HSV model, 4 images were used. These images again were birds' eye, 20°, and 45° angles and all had a shade of grey for the ground colour. For the detection of bays, the same method above was used.

For the occupancy detection the results were as shown in table 2.0.

<i>Total Spaces</i>	<i>Total Vehicles</i>	<i>Image Angle</i>	<i>Space detection (%)</i>	<i>Occupancy detection (%)</i>	<i>Space detection error (%)</i>	<i>Occupancy detection error (%)</i>
28	12	20° Angle	100	92.8	0	7.2
14	1	Birds eye	85.7(12)	85.7	14.3	14.3
8	4	45° Angle	100	87.5	0	12.5
24	8	Birds eye	100	79.1	0	20.9

Table 2.0: Results of space and occupancy detection of car park images at various angles using HSV segmentation method

Summary

Overall, I am happy with the results achieved for both the car park data used for the implementation and the new data. The target results I had set in most cases have been exceeded as illustrated in figures 19.1 and 19.2.

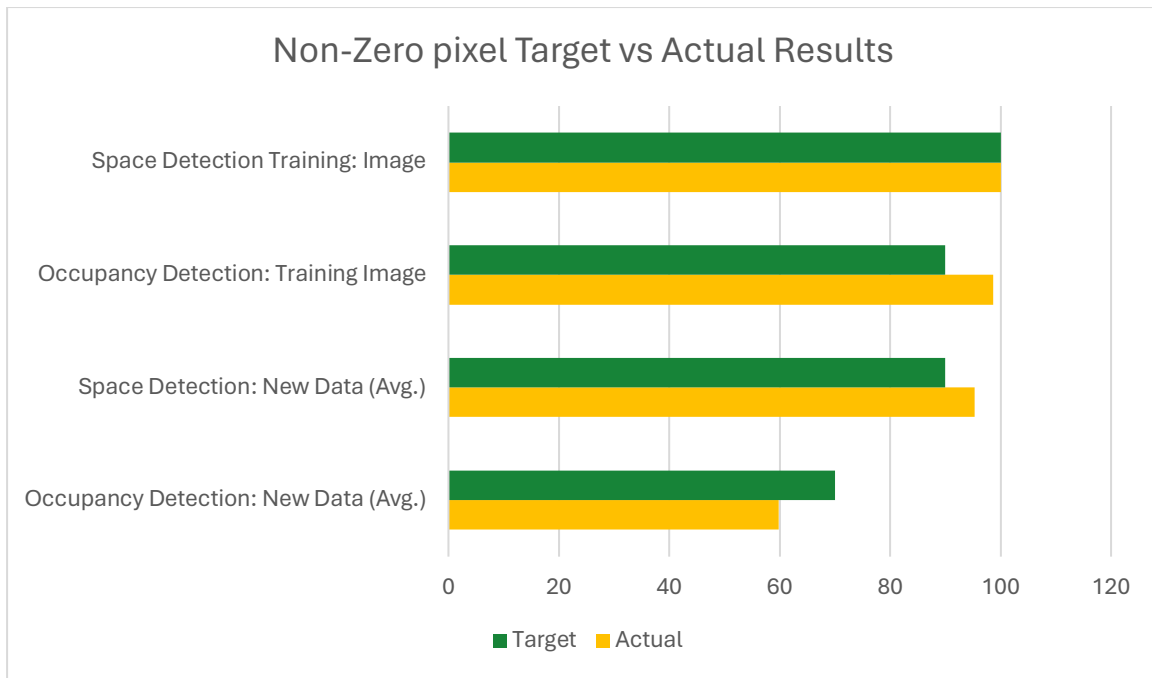


Figure 19.1: Target vs Actual results using non-zero-pixel method.

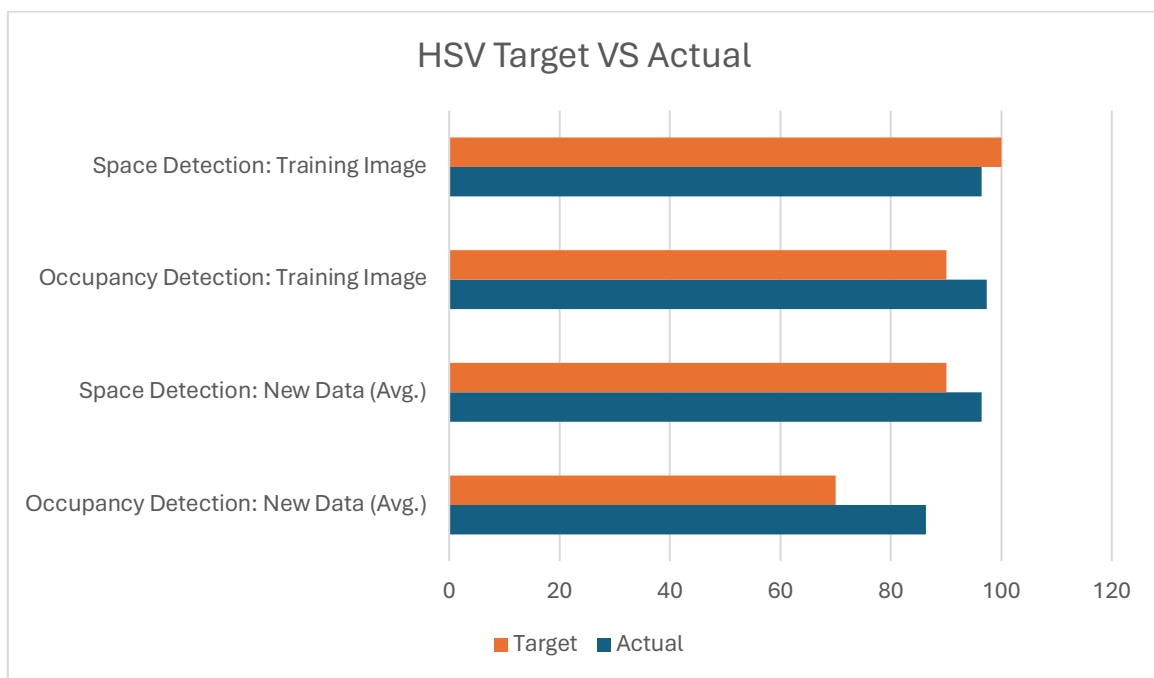


Figure 19.2: Target Vs Actual results using HSV segmentation method.

I think for future iterations what can be improved is finding a better way to highlight the edges of each bay for the segmentation as opposed to manually emphasising each bay's lines. This process was quite tedious hence why only 10 images were used for testing. Using high quality images might have made this process easier unfortunately this was not feasible for the prototype.

When it comes to the bay status detection, although it might seem the HSV model is better, however in the video the non-zero segmentation method had more accurate results which

means in real-time application it would perform better. I believe the optimum solution would be using a pretrained object detection model for detecting cars within the ROI.

11. Conclusions and Recommendations

In conclusion this research project developed a prototype application that has the potential to solve the car parking issue. Image processing has proven to be a valid technique for solving this problem. The backend of the application was developed in Python using the OpenCV library for various image processing and computer vision tasks. A RESTful web service was implemented for communicating with the frontend using the FLASK API Framework. AJAX was used for fetching the required data from the backend and with the use of HTML and CSS the data was displayed in a user-friendly way on a web page.

The target at the initial stage of the project was to achieve 90% success rate for parking space detection which based on the results has been exceeded. For the occupancy detection the target rate was disappointing using the counting non-zero-pixels method, the target was 70% but the actual was 59.75% for still images. Using the HSV segmentation method proved to be a better option for still images with an 86.3% success however for the video some of the bays were fluctuating quite a bit as the video played which might not be ideal for a real-time model.

Future Work

Had there been more time to develop the prototype, there are a few things that could be improved on and added. The main feature requiring improvement is the occupancy detection. The results obtained indicate a better solution is required. Using a pretrained object detection model for detecting cars in the ROI could be a much better solution. For the prototype the focus was more on the backend of the application. Ultimately the application's goal is to eliminate the stress and frustrations drivers feel when looking for a parking space. Therefore, a more interactive and user-friendly frontend should be developed. A better solution would be using a frontend framework such as React Native. These changes and additions should improve the occupancy detection results significantly, make the frontend more interactive and move much closer to a finished product. Other nice to have features could be, a mobile application, payments option (for commercial car parks) and integration with mapping services.

Reflections

Upon reflection on this project and comparing my knowledge between now and at the beginning I am very happy with my development both in theory and practical. Considering I didn't have much of a background in image processing and computer vision, I think the prototype truly is a good start and I am very proud of it. What really helped me develop and understand key concepts is doing the literature review and I am really amazed by how much can be achieved by manipulating images and by using libraries such as OpenCV!

Whilst I am proud of the work I have done and the prototype I have developed, I am disappointed that I did not achieve everything I had planned. I overestimated how many features I could implement for the prototype. This caused me to lose a lot of time looking into theories and tutorials/courses on features that I didn't end up implementing. This time could have been spent on perfecting the features I did implement. I am also gutted that the bay status detection method I used did not end up being as accurate when using images of other car parks. I say this however I am aware I used image processing and the videos/images used are not the highest of quality, so some of the techniques used needed the parameters adjusted to accurately update the statuses in new images.

Overall, I am very happy with my achievements and take the negatives as lessons learnt and aim to continue developing my knowledge in computer vision and machine learning and perfecting the application in my free time.

12. Appendices

- [1] Live Parking Service To Solve Parking Problem of 80% Drivers- <https://www.sygic.com/blog/2015/live-parking-service-to-solve-parking-problem-of-80-drivers>
- [2] Zainal Abidin, Muhammad & Pulungan, Reza. (2020). A Systematic Review of Machine-vision-based Smart Parking Systems. *Scientific Journal of Informatics*. 7. 213-227. 10.15294/sji.v7i2.25654.
- [3] Hemant Warudkar | 04 Oct 2016- <https://www.expressanalytics.com/blog/10-things-about-computer-vision-and-data-analytics-you-may-not-know/> - 10 Ways Computer Vision and Data Analytics Help Your Business: Computer Vision Analytics
- [4] X. Zhang and S. Xu, "Research on Image Processing Technology of Computer Vision Algorithm," *2020 International Conference on Computer Vision, Image and Deep Learning (CVIDL)*, Chongqing, China, 2020, pp. 122-124, doi: 10.1109/CVIDL51233.2020.00030.
- [5] Great Learning Team, What is Image Recognition and How it is Used? Jun 05, 2023, <https://www.mygreatlearning.com/blog/image-recognition/>
- [6] O. A. Abdalla, M. N. Zakaria, S. Sulaiman and W. F. W. Ahmad, "A comparison of feed-forward back-propagation and radial basis artificial neural networks: A Monte Carlo study," *2010 International Symposium on Information Technology*, Kuala Lumpur, Malaysia, 2010, pp. 994-998, doi: 10.1109/ITSIM.2010.5561599.
- [7] Islam, M., Chen, G. and Jin, S., 2019. An overview of neural network. *American Journal of Neural Networks and Applications*, 5(1), pp.7-11.
- [8] N. A. Al-Sammarraie, Y. M. H. Al-Mayali and Y. A. Baker El-Ebiary, "Classification and diagnosis using back propagation Artificial Neural Networks (ANN)," *2018 International Conference on Smart Computing and Electronic Enterprise (ICSCEE)*, Shah Alam, Malaysia, 2018, pp. 1-5, doi: 10.1109/ICSCEE.2018.8538383
- [9] What is a neural network, Amazon Web Services, <https://aws.amazon.com/what-is/neural-network/#:~:text=A%20neural%20network%20is%20a,that%20resembles%20the%20human%20brain.>
- [10] Yamashita, R., Nishio, M., Do, R.K.G. et al. Convolutional neural networks: an overview and application in radiology. *Insights Imaging* 9, 611–629 (2018). <https://doi.org/10.1007/s13244-018-0639-9>
- [11] R. Pramoditha – Overview of a Neural Network's Learning Process - Neural Networks and Deep Learning Course: Part 8 –Feb 1, 2022- date accessed: 28/12/2023 - <https://medium.com/data-science-365/overview-of-a-neural-networks-learning-process-61690a502fa>

- [12] S. Albawi, T. A. Mohammed and S. Al-Zawi, "Understanding of a convolutional neural network," *2017 International Conference on Engineering and Technology (ICET)*, Antalya, Turkey, 2017, pp. 1-6, doi: 10.1109/ICEngTechnol.2017.8308186.
- [13] Phung, & Rhee,. (2019). A High-Accuracy Model Average Ensemble of Convolutional Neural Networks for Classification of Cloud Image Patches on Small Datasets. *Applied Sciences*. 9. 4500. 10.3390/app9214500. https://www.researchgate.net/figure/Schematic-diagram-of-a-basic-convolutional-neural-network-CNN-architecture-26_fig1_336805909
- [14] Kour, Amandeep & Yv, Vi & Maheshwari, Vikas & Prashar, Deepak. (2012). A Review on Image Processing. *International Journal of Electronics Communication and Computer Engineering*. 4. 2278-4209.
- [15] Ruiz-Santaquiteria, J., Bueno, G., Deniz, O., Vallez, N. and Cristobal, G., 2020. Semantic versus instance segmentation in microscopic algae detection. *Engineering Applications of Artificial Intelligence*, 87, p.103271.
- [16] Kirillov, A., He, K., Girshick, R., Rother, C. and Dollár, P., 2019. Panoptic segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 9404-9413)
- [17] C. Liu, X. Lu, S. Ji and W. Geng, "A fog level detection method based on image HSV color histogram," *2014 IEEE International Conference on Progress in Informatics and Computing*, Shanghai, China, 2014, pp. 373-377, doi: 10.1109/PIC.2014.6972360.
- [18] S. T. Rahman, A. Arefeen, S. S. Mridul, A. I. Khan and S. Subrina, "Human Age and Gender Estimation using Facial Image Processing," *2020 IEEE Region 10 Symposium (TENSYP)*, Dhaka, Bangladesh, 2020, pp. 1001-1005, doi: 10.1109/TENSYP50017.2020.9230933.
- [19] M. A. Mir, F. Qazi, M. Naseem, S. S. Zia, D. -e. -S. Agha and T. Mubeen, "Invisibility Cloak using Color Extraction and Image Segmentation with OpenCV," *2022 Global Conference on Wireless and Optical Technologies (GCWOT)*, Malaga, Spain, 2022, pp. 1-6, doi: 10.1109/GCWOT53057.2022.9772883
- [20] J. Brownlee, *A Gentle Introduction to Object Recognition With Deep Learning*, 22/05/2019, <https://machinelearningmastery.com/object-recognition-with-deep-learning/>
- [21] J. U. Kim and Y. Man Ro, "Attentive Layer Separation for Object Classification and Object Localization in Object Detection," *2019 IEEE International Conference on Image Processing (ICIP)*, Taipei, Taiwan, 2019, pp. 3995-3999, doi: 10.1109/ICIP.2019.8803439.
- [22] Y. Jiang, H. Shin and H. Ko, "Precise Regression for Bounding Box Correction for Improved Tracking Based on Deep Reinforcement Learning," *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Calgary, AB, Canada, 2018, pp. 1643-1647, doi: 10.1109/ICASSP.2018.8462063.
- [23] L. Qu et al., "Improving object detection accuracy with region and regression based deep CNNs," *2017 International Conference on Security, Pattern Analysis, and Cybernetics (SPAC)*, Shenzhen, China, 2017, pp. 318-323, doi: 10.1109/SPAC.2017.8304297.

- [24] Subhashini, D., & Dutt, V. S. I. (2020). A review on road extraction based on neural and non-neural networks. *Int J Eng Res*, 9, 1306-1309
- [25] P. Soviany and R. T. Ionescu, "Optimizing the Trade-Off between Single-Stage and Two-Stage Deep Object Detectors using Image Difficulty Prediction," *2018 20th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, Timisoara, Romania, 2018, pp. 209-214, doi: 10.1109/SYNASC.2018.00041.
- [26] Singh, S., 2021. Image filtration in Python using openCV. *Turkish Journal of Computer and Mathematics Education (TURCOMAT)*, 12(6), pp.5136-5143.
- [27] H. Lee and H. Lee, "Average Blurring-based Anomaly Detection for Vision-based Mask Inspection Systems," *2021 21st International Conference on Control, Automation and Systems (ICCAS)*, Jeju, Korea, Republic of, 2021, pp. 2144-2146, doi: 10.23919/ICCAS52745.2021.9649945.
- [28] Siddharth Misra, Yaokun Wu, in *Machine Learning for Subsurface Characterization*, 2020 <https://www.sciencedirect.com/topics/engineering/gaussian-blur#:~:text=where%20%CF%83%20is%20the%20standard,blurring%20effect%20around%20a%20pixel>.
- [29] Ibrahim, N.M., Abou ElFarag, A. and Kadry, R., 2021, April. Gaussian Blur through Parallel Computing. In *IMPROVE* (pp. 175-179).
- [30] Median Filter - Use of a median filter to improve an image severely corrupted by defective pixels – Wikipedia – date accessed: 02/01/2024 – https://en.wikipedia.org/wiki/Median_filter
- [31] Ziou, D. and Tabbone, S., 1998. Edge detection techniques-an overview. *Pattern Recognition and Image Analysis C/C of Raspoznavaniye Obrazov I Analiz Izobrazhenii*, 8, pp.537-559.
- [32] Vairalkar, M.K. and Nimbhorkar, S.U., 2012. Edge detection of images using Sobel operator. *International Journal of Emerging Technology and Advanced Engineering*, 2(1), pp.291-293.
- [33] Abhisek Jana – “How to implement Sobel edge detection using Python from scratch”- May 19, 2019- A Developer Diary – date accessed: 30/10/2023 – <https://www.adeveloperdiary.com/data-science/computer-vision/how-to-implement-sobel-edge-detection-using-python-from-scratch/>
- [34] R. Bakshee - “Edge Detection in Computer Vision” – Feb 3, 2021 - Medium – Date accessed: 30/10/2023 - <https://medium.datadriveninvestor.com/edge-detection-in-computer-vision-6c2c25100bca>
- [35] Song, R., Zhang, Z. and Liu, H., 2017. Edge connection based Canny edge detection algorithm. *Pattern Recognition and Image Analysis*, 27, pp.740-747.

- [36] J. de Oliveira, L. Soares, E. Costa and S. Bampi, "Exploiting approximate adder circuits for power-efficient Gaussian and Gradient filters for Canny edge detector algorithm," *2016 IEEE 7th Latin American Symposium on Circuits & Systems (LASCAS)*, Florianopolis, Brazil, 2016, pp. 379-382, doi: 10.1109/LASCAS.2016.7451089.
- [37] R. K. Sidhu, "Improved canny edge detector in various color spaces," *Proceedings of 3rd International Conference on Reliability, Infocom Technologies and Optimization*, Noida, India, 2014, pp. 1-6, doi: 10.1109/ICRITO.2014.7014744.
- [38] Canny Edge Detection – OpenCv Documentation – Date Accessed: 01/12/2023 - https://docs.opencv.org/3.4/d7/de1/tutorial_js_canny.html
- [39] Martin A, Tosunoglu S. Image processing techniques for machine vision. Miami, Florida. 2000 May 4:1-9.
- [40] P. Roy, S. Dutta, N. Dey, G. Dey, S. Chakraborty and R. Ray, "Adaptive thresholding: A comparative study," *2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)*, Kanyakumari, India, 2014, pp. 1182-1186, doi: 10.1109/ICCICCT.2014.6993140.
- [41] N. V. V. Ravipati, V. R. N. Mondreti, S. K. Satti, C. S. Gurugunti and V. N. S. L. Jakkampudi, "Computer Vision-Based System for Locating and Counting Vacant Parking Lot," *2022 IEEE International Conference on Data Science and Information System (ICDSIS)*, Hassan, India, 2022, pp. 1-6, doi: 10.1109/ICDSIS55133.2022.9915916.
- [42] Srisha, Ravi & Khan, Am. (2013). Morphological Operations for Image Processing: Understanding and its Applications.
- [43] *Dilation* – date accessed: 08/12/2023 - <https://homepages.inf.ed.ac.uk/rbf/HIPR2/dilate.htm>
- [44] *Erosion* – date accessed: 08/12/2023 - <https://homepages.inf.ed.ac.uk/rbf/HIPR2/erode.htm>
- [45] Brian Thorne - University of Canterbury- Introduction to Computer Vision in Python - The Python Papers Monograph, Vol. 1 (2009)
https://www.researchgate.net/publication/228378315_Introduction_to_Computer_Vision_in_Python
- [46] I. Culjak, D. Abram, T. Pribanic, H. Dzapo and M. Cifrek, "A brief introduction to OpenCV," *2012 Proceedings of the 35th International Convention MIPRO*, Opatija, Croatia, 2012, pp. 1725-1730.
- [47] OpenCV Documentation – About – date accessed 02/01/2025 - <https://opencv.org/about/>

- [48] J.Das, M.Pulipalupula, S.Patlola, M.Nayaki, M.Yadlapati, B.R.Sanjeeva Reddy, "Object Detection using You Only Look Once (YOLO) Algorithm in Convolution Neural Network (CNN)", , 2023 IEEE 8th International Conference for Convergence in Technology (I2CT) Pune, India. Apr 7-9, 2023
- [49] JR.Terven, D.M. Córdova-Esparza, A COMPREHENSIVE REVIEW OF YOLO: FROM YOLOV1 TO YOLOV8 AND BEYOND, Apr 4, 23.
- [50] A. Tripathi, M. K. Gupta, C. Srivastava, P. Dixit and S. K. Pandey, "Object Detection using YOLO: A Survey," *2022 5th International Conference on Contemporary Computing and Informatics (IC3I)*, Uttar Pradesh, India, 2022, pp. 747-752, doi: 10.1109/IC3I56241.2022.10073281.
- [52] Kadir, Muhamad & Osman, Mohd & Othman, Nor & Sedek, Khairul Anwar. (2020). IoT based Car Parking Management System using IR Sensor. *Journal of Computing Research and Innovation*. 5. 75-84. 10.24191/jcrinn.v5i2.151.
- [53] Al-Absi, Hamada & Devaraj, Justin & Sebastian, Patrick & Yap, Vooi. (2010). Vision-based automated parking system. 757 - 760. 10.1109/ISSPA.2010.5605408.
- [54] E.Akkharawat , "Real-time vehicle detection and counting using Background Subtractor comparing with Haar Cascade Classifier", Medium, Dec 06, 22, date accessed: 10/01/2024 - <https://medium.com/@EarthAkkharawat/real-time-vehicle-detection-and-counting-using-the-background-subtractor-comparing-with-haar-ff05f9ab2b92>
- [55] OpenCV Documentation, Contours: Getting Started, https://docs.opencv.org/3.4/d4/d73/tutorial_py_contours_begin.html, Date Accessed: 30/10/2023.
- [56] OpenCV Documentation, Contours Hierarchy, https://docs.opencv.org/4.x/d9/d8b/tutorial_py_contours_hierarchy.html, Date Accessed: 30/10/2023.
- [57] OpenCV Documentation, Countour Features, https://docs.opencv.org/4.x/dd/d49/tutorial_py_contour_features.html. Date Accessed: 30/10/2023.
- [58] pickle Python Object Serialization- <https://docs.python.org/3/library/pickle.html>, Date Accessed: 17/11/2023.
- [59] GeeksForGeeks – Understanding Python Pickling with example –27 Oct, 2023 - Date accessed: 17/11/2023 - <https://www.geeksforgeeks.org/understanding-python-pickling-example/>
- [60] Numpy – numpy.all - <https://numpy.org/doc/stable/reference/generated/numpy.all.html> - Date accessed: 22/01/2024.