Robot Digital Augmented Reality View

Presented by:
S'thabiso Samkelo Lushaba
LSHSTH002

Prepared for:
Simon Winberg
Dept. of Electrical and Electronics Engineering
University of Cape Town

Submitted to the Department of Electrical Engineering at the University of Cape Town in partial fulfilment of the academic requirements for a Bachelor of Science degree in Electrical and Computer Engineering.

May 20, 2024

Key words: Computer Vision, Object detection, Remote navigation, Augmented Reality

Declaration

1. I know that plagiarism is wrong. Plagiarism is to use another's work and pretend that it is one's own.
2. I have used the IEEE convention for citation and referencing. Each contribution to, and quotation in, this report from the work(s) of other people has been attributed, and has been cited and referenced.
3. This report is my own work.
4. My use of artificial intelligence software has been limited to grammar corrections, latex code structures for making tables, and overall organizing the document and summarizing of longer texts during the research phase to help get an understanding of the paper before reading it, this helped with saving time. (specify precisely how you used AI to assist with this assignment)
5. I have not allowed, and will not allow, anyone to copy my work with the intention of passing it off as their own work or part thereof.

Signature:.....S.S Lushaba.....
S'thabiso Samkelo Lushaba

Date: May 20, 2024

Acknowledgments

I extend my deepest gratitude to Professor Simon Wingerg for his exceptional supervision and guidance throughout this project, my family for their unwavering support throughout this journey. Additionally, I am immensely grateful to my friends for their unwavering support and encouragement. Special thanks to Bradley and Thomas, I could go on for forever bout times y'all came through, shout outs. To all my friends, near and far, your presence in my life has been invaluable, and I am thankful for each of you. Msizi, we will definitely continue Mkhandlu in a different setting with even crazier content.

I would like to express my gratitude to everyone who has positively impacted my life and crossed paths with me along the way. I want to acknowledge my sister for her unwavering support and everything she has done for me. Your presence in my life has been a source of strength and inspiration. Growing up with you has been nothing short of amazing.

Lastly, I want to thank myself for standing on business, persevering through the challenges and staying committed to achieving my goals. Despite the uncertainties along the way, I remained determined for to get this far.

Abstract

This project explores the integration of augmented reality (AR) into remote navigation systems, aiming to enhance the real-time environmental perception and interaction capabilities of autonomous robots. A versatile robot kit was constructed using Arduino, complemented by a Raspberry Pi and its camera module. This hardware setup provides a robust platform for capturing live footage and performing real-time object detection, forming the basis for advanced navigation functionalities. Contrasting with virtual reality (VR), which creates entirely immersive digital environments, AR overlays digital information onto the real world, augmenting the user's perception without isolating them from their physical surroundings. This project leverages AR to provide enhanced situational awareness and interaction capabilities for remote navigation.

The implementation of object detection algorithms using the YOLOv8 model proved successful, with the system reliably identifying and classifying objects in real-time with high accuracy. The model, trained on a custom dataset, demonstrated proficiency in recognizing target classes, significantly contributing to the system's overall perceptual capabilities. However, challenges were encountered during the object counting process, particularly in dynamic environments. These difficulties highlight the need for further refinement of counting algorithms to ensure accurate and reliable performance in real-world scenarios.

The findings from this project highlight the transformative potential of AR in enhancing remote navigation systems. Despite the challenges faced, particularly with object counting, the integration of AR with real-time object detection demonstrated significant advancements in navigation and environmental interaction capabilities. This project underscores the importance of continuous innovation and improvement in robotics and AR, paving the way for more advanced and reliable remote navigation systems.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background to the study

Industry 4.0[18] has seen the integration of robotics and augmented reality (AR) technologies, emerging as a promising field with diverse applications ranging from industrial automation[19] to healthcare[20] and education[21]. Augmented reality, alongside robotics, is revolutionizing how we interact with machines and the environment around us. By overlaying digital information onto the physical world, AR enables robots to perceive and interact with their surroundings more effectively. This integration not only improves efficiency but also enhances safety and decision-making processes and with AI-powered algorithms, comes the ability for robots to interpret visual data in real-time, allowing for autonomous decision-making and adaptive navigation. This empowers robots to navigate complex environments remotely, opening up new possibilities for applications such as remote inspection, surveillance, and exploration.

Overall, the integration of augmented reality, robotics, and artificial intelligence offers exciting opportunities for remote navigation and interaction. As we continue to advance in Industry 4.0, the synergy between these technologies will drive innovation across various sectors, revolutionizing how we perceive and interact with the world around us.

1.2 Objectives of this study

1.2.1 Problems to be investigated

As recent advancements in computer vision technology become increasingly accessible and user-friendly, researchers, enthusiasts, and others can engage with these technologies through various projects of differing complexities. This paper addresses the primary challenges of designing a system aimed at enabling smart remote navigation. While video feed from a camera attached to a robot kit is essential for exploring the environment, it requires real-time monitoring to observe changes in the surroundings.

This study aims to address this limitation by exploring different computer vision algorithms and technologies capable of augmenting the reality. This means reading, analyzing, and extracting information from the robot's surrounding environment. This approach aims to eliminate the need for manual tracking and environmental analysis, as these tasks can be performed in real-time, faster, and with greater accuracy using these technologies, even in diverse environments. For instance, a system equipped with night vision cameras would still deliver optimal performance.

1.2.2 Purpose of the study

This study aims to develop a system featuring a remotely controlled 4-wheeled robot equipped with a securely attached camera to capture video feed of its surroundings. The captured feed is then transmitted to a central host computer or command center where, alongside remote navigation controls, real-time processing of the feed is executed. The system seeks to augment the robot-camera feed through exploration of the computer vision field. Subfields such as object detection and tracking are investigated to fulfill the system requirements. These requirements encompass the development of a user-friendly interface enabling users to toggle the augmentations on or off as needed. Among the targeted augmentations are object detection and the enumeration of instances belonging to a predefined class within the robot's visual field. By leveraging advancements in computer vision technology, the purpose is to enhance the robot's perception and navigation abilities, ultimately contributing to the development of smarter and more efficient robotic systems for various applications.

1.3 Scope and Limitations

Introducing computer vision into a robot for remote navigation is justified when the robot operates beyond the controller's line of sight, a scenario typically facilitated by Radio Frequency (RF) communications. However, due to resource availability and the sample size of this project, the implementation is limited to Bluetooth communication. Given the time constraints of 11 weeks, developing object detection models from scratch was deemed unfeasible. Instead, the project explored open-source algorithms, encompassing both pre-trained and custom-trained models. The development of the app used to control the robot from a cellphone was considered outside the project's scope. Variations in lighting conditions were acknowledged as a potential challenge, which could lead to decreased accuracy or reliability in real-world scenarios. Additionally, factors such as hardware limitations, processing power, and environmental conditions (e.g., obstacles, terrain) may impact the overall performance of the system but were not extensively investigated within the scope of this project.

1.4 Plan of development

BeginningBeginning with a literature review in Chapter 2, the remainder of the project is structured as follows. In Chapter 3, the methodology section details the approach taken to develop and deploy the model responsible for object detection and tracking, including the selection of algorithms and hardware components. Chapter 4 focuses on the system design, outlining the architecture and integration of the augmentations into the robot. Included also in this chapter is the architecture design of the robot. Following implementation, Chapter 5 presents the results obtained from testing and evaluating, assessing the system's performance in various scenarios. In Chapter 6, the discussion analyzes the findings, and the relevance of these result. Chapter 7 draws conclusions based on the project outcomes, summarizing key findings and their implications. Finally, Chapter 8 offers recommendations for future research and potential enhancements to the system. Figure 1.1 provides a graphical presentation of the plan of development.

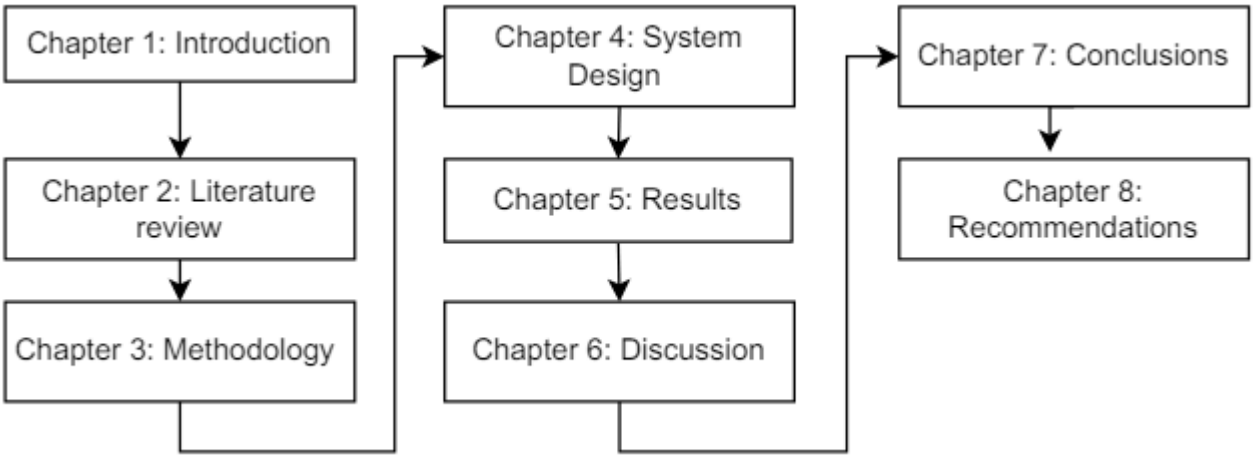


Figure 1.1: Plan of development flow chart

Chapter 2

Literature Review

2.1 Computer Vision : A Brief Overview

Computer Vision, as its name suggests, is a branch of artificial intelligence (AI) dedicated to empowering computer systems to extract insights from provided visual data. This visual data, often sourced from live camera inputs, images, videos, and more, undergoes processing and interpretation by computational algorithms [22]. This enables various systems to gain an awareness of their surroundings, akin to human perception but on a significantly more advanced level. As the scale and complexity of these systems increase, the breadth of collected data knows no bounds, further enhanced by technologies such as night vision, among others. Object detection, an important subclass of computer vision centered on the identification of specific instances of visual objects within digital images or videos, including humans, animals, or cars [1], forms the basis for computer vision. Its objective is to develop computational models necessary for supplying fundamental knowledge to a variety of computer vision applications, such as augmented reality AR. Following is a brief history of object detection and its current state as the fundamental element of computer vision.

2.1.1 Object Detection: History

Over the past two decades, the evolution of object detection has been categorized into two distinct historical periods: traditional object detection, which persisted until approximately 2014, and the subsequent era dominated by deep learning-based detection methodologies

[1]. Numerous methods/algorithms emerged within these periods, with origins dating back to the 1990s as shown in figure 2.1.

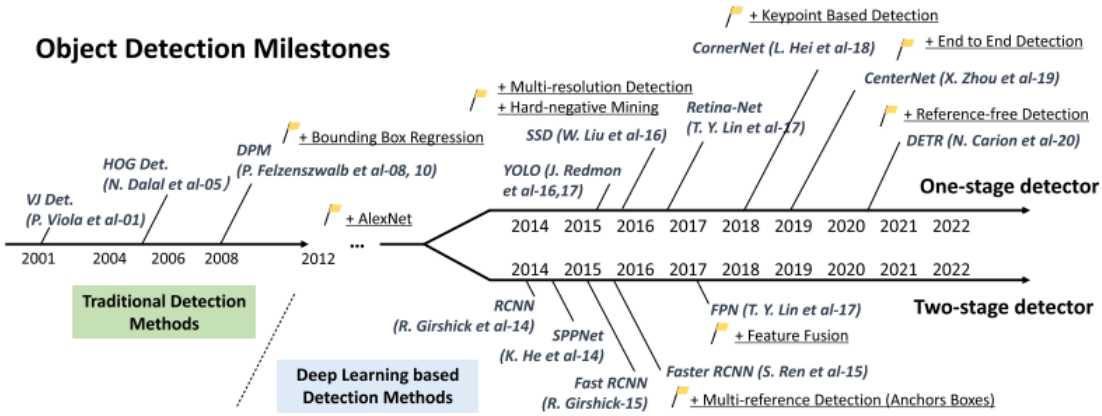


Figure 2.1: Object detection milestones over the years. Adapted from [1]

Herein is a concise overview of some algorithms that have gained prominence over the years and are now widely utilized in contemporary technological endeavors employing computer vision, as have open-source implementations available.

1. **Histogram of Oriented Gradients (HOG)**

   HOG, introduced in 1986 by Robert K. McConnell, stands as one of the earliest methods of object detection [23]. Although the term "HOG" wasn't coined until the 2000s, particularly 2005 when it would gain its popularity. Robert introduced the concept in a patent application for an invention related to pattern recognition. This patent focused on methods and apparatuses for analyzing patterns, both static and dynamic, and applying such methods to control processes. The idea revolves around characterizing local objects and shapes by the distribution of local intensity gradients-the change in the direction of the intensity level of an image or edge directions, even without knowing the corresponding gradient or edge positions [3]. This is achieved by dividing the image into small sections called cells, where histograms showing how colors change or edges appear are created. These histograms from the cells are then combined to represent the full image as can be seen in figure 2.2.

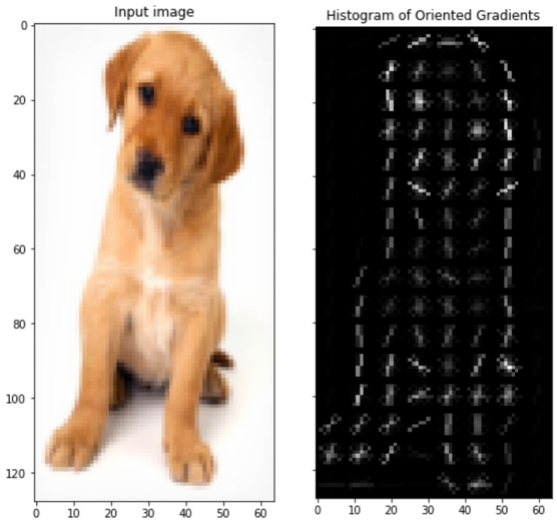


Figure 2.2: Histograms of oriented gradients of a dog input image.
adapted from[2]

To provide robustness against local variations in illumination and contrast, cells are grouped into larger blocks, and histograms from these blocks are concatenated to form a feature vector. This feature vector effectively encodes information about the local edge or gradient structure within the image[3]. Figure 2.3 below presents an overview example of the use of HOG for human detection presented by Navneet Dalal and Bill Triggs, in which they concluded that utilizing locally normalized histogram of gradient orientations features akin to SIFT descriptors within a dense overlapping grid yields very promising results for person detection [24]. Their approach significantly reduces false positive rates by more than an order of magnitude relative to the best Haar wavelet-based detector referenced in their study [25].

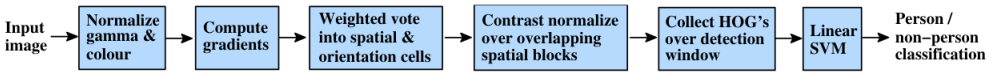


Figure 2.3: HOG for human detection system overview .
adapted from[3]

One modern application of HOG is in pedestrian detection systems used in autonomous vehicles or surveillance systems. By analyzing the distribution of edge orientations in different parts of an image, HOG-based detectors can effectively identify the presence of pedestrians even in cluttered or crowded scenes.
For instance, in an urban traffic monitoring scenario, HOG features can be extracted from video frames to detect pedestrians near roadways, helping autonomous vehicles navigate safely through complex environments. [26]

2. Region-based Convolutional Neural Networks (R-CNN)

Introduced by Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik in 2014. R-CNN is an extension of traditional Convolutional Neural Networks (CNNs) designed specifically for the task of object localization and detection.[1].CNNs are specialized neural networks that effectively processes input data using convolutional operations to recognize key features. CNNs are widely employed for accurately identifying elements within images. To address the issue of selecting numerous regions, R-CCN utilizes selective search to extract only 2000 regions from the image, termed as region of interest (ROI). This approach enables focusing on a manageable set of regions for classification. Figure 2.4 below shows an overview of R-CNN for object classification.

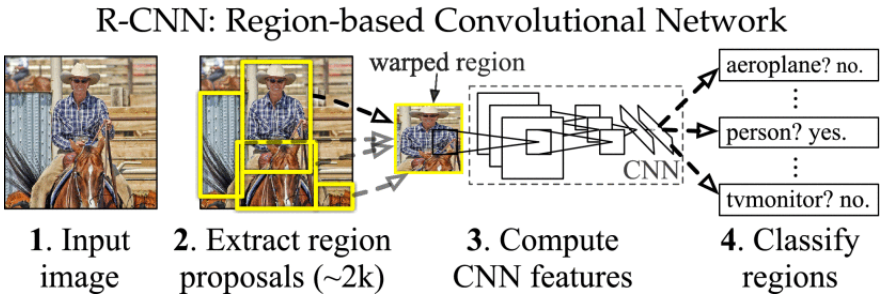


Figure 2.4: R-CNN object detection system overview. Adopted from [4]

In figure 2.4, The process begins with the conversion of 2000 candidate region proposals into square shapes, followed by their input into a CNN, resulting in a 4096-dimensional feature vector extraction. Acting as a feature extractor, the CNN yields dense features, which are then passed through a Support Vector Machine (SVM) for object presence classification within each proposal. Beyond mere object detection, the algorithm predicts four offset values to enhance bounding box precision. This refinement step addresses scenarios where detected objects may be partially occluded within the proposal, ensuring more accurate localization. R-CNN however has its downsides, as it is costly to train the network, considering 2000 ROI per image. As a result R-CNN cannot be implemented in real time. Another major issue regarding R-CNNs is that because selective search is a fixed algorithm, it doesn't adapt or learn from the data it encounters during training. As a result, it may generate ROIs that are not optimal or fail to capture all relevant objects in an image. These downsides would lead to a short lived life of R-CNN as can be seen in figure2.1. Shortly after fast R-CNN and faster R-CNN would come to life, herein is a discussion of the two.

(a) **Fast R-CNN**

The following year after the introduction of R-CNN, Ross Girshick would take note of R-CNN drawbacks; As above mentioned, R-CNN training is expensive in space and time. To prepare for SVM and bounding-box regressor training, the algorithm sifts through each image's object proposals, extracting features and saving them onto disk. For extensive networks like VGG16[27], this operation devours 2.5 GPU-days to handle the 5k images within the VOC07 trainval set. The sheer volume of these features demands hundreds of gigabytes of storage capacity[5]. The detection also happens slower than desirable, taking 47 seconds/image with VGG16 on GPU. In his paper, he would introduce Fast R-CNN to do away with the downsides of R-CNN. Figure 2.5 below illustrates the fast R-CNN architecture.

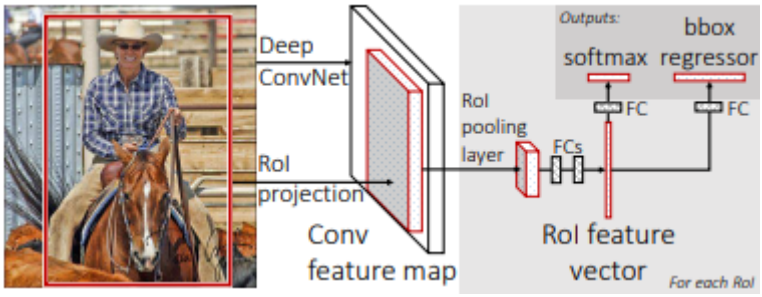


Figure 2.5: Fast R-CNN architecture. Adapted from [5]

As can be seen in figure 2.5, Fast R-CNN revolutionizes the traditional R-CNN by taking the entire image as input and a set of object proposals, streamlining the process. Instead of individually processing each proposal, Fast R-CNN initially evaluates the entire image with convolutional and max pooling layers to generate a feature map. Subsequently, it utilizes a ROI pooling layer to extract fixed length feature vectors from the feature map of each proposal. This integration significantly accelerates processing speed and enhcances efficiency compared to the sequential approach of R-CNN, which involves redundant feature extraction for each proposal.

(b) **Faster R-CNN**

In the year 2016, as further research and other object detection algorithms were coming out in numbers, Shaoqing Ren et al introduced an even faster version of the R-CNN, called Faster R-CNN which is a two module object

detection system. Shaoqing Ren et al. described, "The first module is a deep fully convolutional network that proposes regions, and the second module is the Fast R-CNN detector that uses the proposed regions"[6]. The Region Proposal Network (RPN), when provided with an input image, generates a series of rectangular object proposals accompanied by scores indicating the probability of each proposal representing an object. Unlike the previous version of R-CNN, which relied on a fixed selective search algorithm to generate region proposals, the RPN predicts these proposals dynamically. This approach leads to optimal generation of ROIs and reduces redundancy in object detection within an image. The introduction of learning of ROIs may seem to overall make the system computationally more expensive and time consuming, however Shaoqing Ren et al described the system to be a single, unified network as can be seen in figure 2.6.

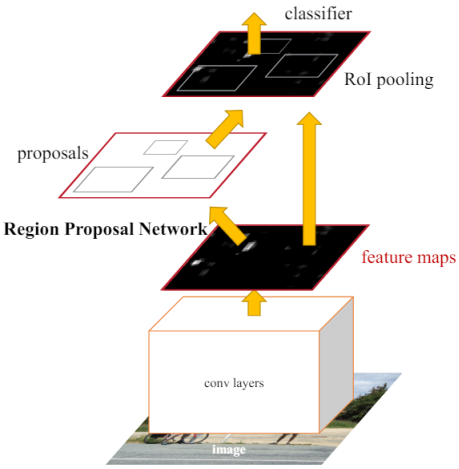


Figure 2.6: Faster R-CNN architecture. Adapted from [6]

With Faster R-CNN, the feature extraction and region proposal stages from R-CNN and Fast R-CNN is combined into a single unified CNN. This streamlined approach offers computational efficiency beyond what either method achieves individually. It also uses an anchor box mechanism to handle multiple scales and aspect ratios, which can improve the robustness of object detection [6]

In conclusion, represents a significant advancement in object localization and detection tasks by leveraging CNNs tailored for this purpose. Despite its effectiveness in reducing false positive rates through selective search and feature extraction, R-CNN's drawbacks, including high computational costs and a fixed selective search algorithm, spurred the development of faster alternatives. Fast R-CNN, in response to these limitations, revolutionized object detection by integrating the entire image

and object proposals into a unified CNN architecture. By eliminating redundant feature extraction and leveraging ROI pooling, Fast R-CNN significantly enhanced processing speed and efficiency compared to its predecessor. Faster R-CNN, a further evolution of the R-CNN framework , dynamically predicting region proposals using a RPN achieved optimal generation of ROIs and reduced redundancy in object detection. Additionally, its integration of feature extraction and region proposal stages into a single unified CNN, along with the use of anchor boxes, further improved computational efficiency and robustness in object detection. While R-CNN laid the foundation for modern object detection systems, Fast R-CNN and Faster R-CNN represent significant advancements that address its limitations, paving the way for more efficient and accurate object detection algorithms.

3. **YOLO (You Only Look Once)**

The computer vision community was introduced to YOLO through a paper published in 2015 by Joseph Redmon et al titled "You Only Look Once: Unified, Real-Time Object Detection" [7]. They noted the downsides of the then existing object detection systems which repurposed classifiers for detection. These systems detected object by taking the classifier of that object and evaluating it across the test images. Models like RCNN2 minimized the computational costs associated with that approach by making use of only proposed regions as opposed to sliding windows at evenly spaced pixel locations approach much like the deformable parts models(DPM) [28].

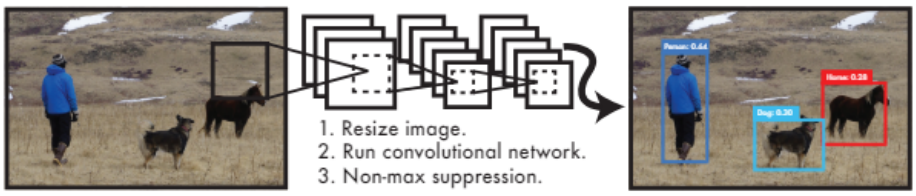


Figure 2.7: The YOLO Detection System overview. Adapted from [7]

In the figure 2.7, Redmon et al showed the system overview of their YOLO model. The model resized all of the input images to 448 × 448, then pass on the images onto a single convolutional network followed by the last stage which involves thresholding the resulting detections by the model's confidence. Redmon explained, "We reframe object detection as a single regression problem, straight from image pixels to bounding box coordinates and class probabilities" [7]. The use of a single CNN that simultaneously predicts bounding boxes and the class probability completely reframes object detection into a single regression probability problem [7]. As such, does not require complex

pipelining, with the base network running at 45 fps on a Titan X GPU [29], achieving 150 fps on a faster version[7]. This allowed for processing streaming video in real-time with as little latency as 25 milliseconds which was unheard of at the time. Since its inception, YOLO has remained one of the most popular and widely used object detection methods. Figure 2.8 illustrates the various versions of YOLO developed over the years, demonstrating its ongoing relevance and evolution with changing times.

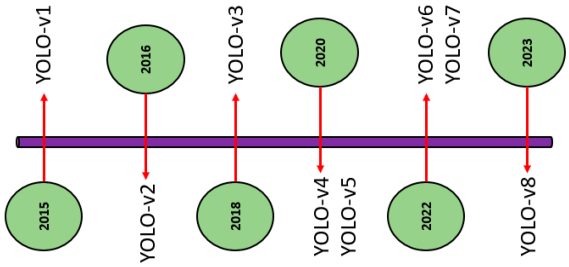


Figure 2.8: YOLO evolution timeline. Adapted from [8]

The fundamental concept introduced by YOLO-v1 (You Only Look Once) involved dividing the image into a grid of cells, each with dimensions s × s as shown in figure 2.7. Within the network, If the center of an object of interest fell within a specific grid cell, that cell would take responsibility for detecting that object. This approach allowed other cells to ignore the object, even if it appeared in multiple grid cells [8]. The imposed grid cells through computational algorithms predict B bounding boxes, dimensions and confidence scores indicative of the absence or presence of an object within the boxes. Each bounding box consists of five components: x, y, w, h, and the confidence score. The first four components represent the center coordinates (x, y) and dimensions (width and height) of the respective bounding box, as shown in Figure 2.9.

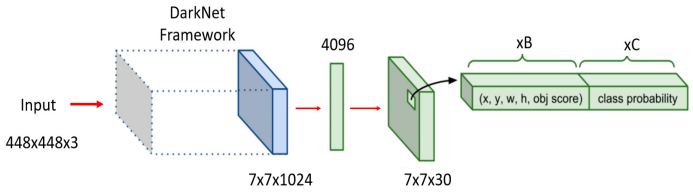


Figure 2.9: YOLO-v1 architecture overview. Adapted from [8]

The primary goal is to accurately identify and localize objects within an image using bounding boxes. This task involves the utilization of two sets of bounding box vectors: the ground truth vector (y) and the predicted vector ($\dot{y}$). In managing scenarios where multiple bounding boxes correspond to the same object or contain

no object, YOLO (You Only Look Once) employs a technique known as non-maximum suppression (NMS). By establishing a threshold value for NMS, overlapping predicted bounding boxes are filtered out, with one of them being discarded if the Intersection over Union (IoU) between two bounding boxes falls below the defined NMS threshold [8]. YOLO-v1 as depicted in figure2.9was initially developed based on the Darknet framework and featured two sub-variants. The first architecture comprised 24 convolutional layers, concluding with a connection to the first of two fully connected layers. The 'Fast YOLO' variant, on the other hand, employed nine convolutional layers, each equipped with fewer filters. Drawing inspiration from GoogLeNet's [30] inception module, YOLO integrated a sequence of 1×1 convolutional layers to diminish the feature space from preceding layers. YOLO leverages a combination of bounding boxes, NMS, and deep learning architecture to achieve efficient and accurate object detection in images. It is however with its limitations, Redmon et al noted that YOLO's bounding box predictions are tightly constrained spatially, as each grid cell is restricted to forecasting just two boxes, each assigned to a single class. This constraint significantly restricts the model's ability to detect nearby objects, particularly challenging its performance with small objects occurring in clusters, like flocks of birds [7]. In essence, it was concluded that the main source of YOLO's errors stems from incorrect localizations [7]. Several companies and research groups have adopted the YOLO models for various computer vision applications, inluding

DeepVisionAI, where they offer services centered around security such as facial recognition license plate Recognition Intelligent Video Analytics, audio analytics etc [31]

2.2 Augmented Reality(AR) Technology

2.2.1 Overview and brief history

Augmented Reality(AR)technology encompasses various forms of technology that enhance the perception of the physical real world. This technology overlays the real-world view with computer-generated information, which can include text, images, videos, and more, hence the term 'augmented' reality. The computer-generated information is fundamentally derived from the real-world view captured by a computer. Through computer vision techniques, the computer generates or extracts information from the captured view in a manner that aligns and enhances both layers of information, resulting in an enhanced

view of the real world [32].
In the 1960s, several groundbreaking technologies emerged, each contributing significantly to the technological landscape we know today [33]. Among these innovations was the Heads-Up Display (HUD) technology, originally developed for military aviation purposes. The earliest HUDs were rudimentary systems that primarily displayed essential flight information onto a transparent screen in the pilot's line of sight[34]. These early HUDs projected data such as airspeed, altitude, navigation information, and targeting data onto a transparent screen in the pilot's line of sight, which allowed for improved situational awareness, enabling pilots to access critical data without diverting their gaze from the external environment.
Over time, HUD technology evolved, finding applications beyond military aircraft into commercial aviation, automotive etc[35]. The concept of overlaying digital information onto the real-world environment, as demonstrated by HUD technology, laid the groundwork for the development of augmented reality. In 1968, associate professor Ivan Sutherland alongside his students Bob Sproull, Quinton Foster, Danny Cohen, and others developed the inaugural head-mounted display. This device rendered images responsive to the viewer's dynamic pose, marking the creation of the initial (AR) system, famously known as "The Sword of Damocles" [34].
AR technology has witnessed substantial progress in tandem with the evolution of everyday technologies as can be seen in figure 2.10 and 2.11. Just in recent years the U.S. Army took delivery of its inaugural batch of Integrated Visual Augmentation System(IVAS) augmented reality combat goggles, marking a significant milestone in military technology integration. These cutting-edge goggles, part of the IVAS system, empower soldiers with the capability to access and exchange data seamlessly among themselves, fostering enhanced situational awareness and collaboration on the battlefield. Moreover, soldiers equipped with IVAS goggles can observe their surroundings in real-time, including external views from vehicles, further augmenting their tactical prowess. However, despite its promising features, the development of the IVAS system, which is rooted in Microsoft's Hololens technology, has been marred by delays and budget overruns, contributing to its tardy deployment[10].

Figure 2.10: Ivan Sunderland's 'The sword of Damocles' prototype
adapted from [9]

Figure 2.11: Microsoft IVAS military goggles
adapted from [10]

2.2.2 Nature of AR

The classification of augmented reality (AR) systems, which is intricately linked to the modality through which augmentation is instigated[36] comprises four types. Presented herein is a concise overview of the diverse AR systems contained within their respective categorization.

1. **Marker-based AR**

   Marker-based AR, also known as image recognition/recognition-based AR[37] relies on the use of a specific marker to trigger the augmentation. These markers can take the form of paper-based patterns or tangible objects in the real world. The augmentations linked to these markers often enhance the visual representation or unique features of the designated image or object [36] as can be seen in figure 2.12 below. A classic example of marker-based AR is the use of Quick Response code(QR) to trigger augmentations, For instance, scanning a QR code next to a diagram of a plant cell could activate an AR simulation that allows users to explore and interact with various organelles within a cell in a 3D environment.

Figure 2.12: Hong Kong Tourism App with marker based AR. Adapted from [11]

Figure2.12 shows an example of marker-based AR technology in a tourism application. The Tourism Commission in partnership with City University of Hong Kong launched CITY IN TIME, a project enabling visitors and residents of Hong Kong to experience the city's past through immersive AR technology at 13 locations in Central and Tsim Sha Tsui. Led by Professor Jeffrey Shaw, the project merges art and technology, showcasing historical scenes alongside present-day views using a smartphone app. Developed by CityU's School of Creative Media, CITY IN TIME offers users the ability to compare past and present views, explore landmarks, and delve into historical information. Through extensive collaboration with historians and artists, the project creates a vivid blend of real surroundings and virtual content, aiming to deepen understanding and appreciation of Hong Kong's cultural heritage. Gradually rolling out across different districts, CITY IN TIME also offers remote access via a dedicated website [11].

2. **Location based AR**

Location-based AR relies on GPS, Wi-Fi, or other location-tracking technologies to detect a user's position and orientation in the real world. Based on the user's location, relevant virtual content is superimposed onto the physical environment. This type of AR allows for contextually relevant information or experiences to be delivered based on the user's geographic location[36]. One prominent example is the application called "Pokémon GO" developed by Niantic [38]. In Pokémon GO, players use their smartphones to explore their real-world surroundings while encountering virtual Pokémon characters overlaid onto the physical environment. These virtual creatures appear in different locations based on the player's geographical coordinates. Figure 2.13 below shows the Pokémon GO outlook.

Figure 2.13: Pokémon GO by Niantic with Augmented Reality. Adapted from [12]

As can be seen in figure 2.13, users explore the real world with their smartphones and the AR application imposes on to their smartphone world view based on the users geographical location.

3. **Projection Based/Spatial AR**.
   In this type of AR, augmentations are projected onto the physical world using projection technology. Additionally, sensors are employed for tracking, as AR applications must adapt to the potentially changing user's point of view [39]. This form of AR has found significant success in various industries such as entertainment, marketing, and system modeling. For instance, in 2010, Disney utilized projection-based AR to reimagine the adventures of Snow White, enhancing the storytelling with detailed scenic projections that couldn't have been realized in the 1930s [40], see figure 2.14.

Figure 2.14: Projection-Bases AR
in Disney Theme Parks. Adapted from [13]

In the figure 2.14, the top left image is the raw image, on the left is the image with projections to augment the reality. As can be seen, the projections enhance the entire view by introducing the correct shading and coloring which at the time of the raw image could not be achieved in animation.

4. **Superimposition Based AR**

In this classification of AR, real-world objects captured by the AR system are either entirely or partially substituted with virtual content [41]. It's essential to highlight that complete replacement of the entire view with virtual content would align more closely with Virtual Reality (VR) technology. VR immerses users in a wholly virtual environment, disconnecting them from reality, while AR maintains the user's connection to the real world [32]. Applications such as Instagram, TikTok, and Facebook utilize this type of AR with real-time filters. For instance, when a user's face is detected by the camera, these applications employ image recognition to substitute the user's eyes with different ones, creating an illusion of some sort. Figure 2.15 shows an example of Superimposition based AR.

Figure 2.15: Superimposition based AR. Adapted from [14]

Figure 2.15 shows virtual content complimenting a historical site for a history field trip imposed by superimposition based AR.

Augmented Reality (AR) is implemented across various platforms, from smartphones and tablets to dedicated AR glasses and web-based experiences. Platforms like Apple's ARKit and Google's ARCore offer tools for smartphone AR development, while devices like Microsoft's HoloLens provide proprietary ecosystems for AR glasses. Web-based AR [42], enabled by platforms like WebXR, offers accessibility through web browsers. Mixed Reality platforms such as Unity's MARS support immersive experiences blending virtual and physical elements. While some components may be open source, the proprietary nature of hardware limits full openness. However, efforts to promote open standards continue to drive innovation in the AR ecosystem

2.3 Robot Kits and Remote Navigation

2.3.1 Overview and history

In 1920, the world would be introduced to the word 'robot' in a play written by Karel Čapek, titled Rossum's Universal Robots(RUR) [43], drawing the word from an old Church Slavonic word, robota, translating to "servitude","forced labor" [44]. This would be no further from the truth, almost two decades later the world would see the use of these robots in warfare during the world war II, except in the form of the German Goliath demolition vehicles or the Soviet teletanks, see figure 2.16. The German demolition vehicles operated remotely, served the Wehrmacht during World War II. Its purpose was to dismantle tanks, scatter dense infantry formations, remove minefields, and raze buildings, all while safeguarding soldiers' lives [15].

Figure 2.16: A Goliath displayed at the Museum of Armored Vehicles in Saumur, France.Adapted from [15]

In contrast to the early conception of robots as depicted in Rossum's Universal Robots, contemporary society witnesses a burgeoning fascination with robotics manifested in various forms, including robot kits readily available for enthusiasts and educational purposes. These kits enable exploration into robotics, encouraging creativity, innovation, and tech skills. With readily available kits featuring sensors, motors, and programmable parts, users can craft their own robots. This reflects humanity's ongoing fascination with automation and AI, moving beyond historical perceptions of robots solely as tools for servitude or warfare. This literature is limited within the context of 4 wheeled robot kits for remote navigation. This literature focuses solely on 4-wheeled robot kits designed for remote navigation.

2.3.2 Architecture

The architecture of 4-wheeled robot kits for remote navigation typically encompasses several fundamental components. These include the chassis, motors, wheels, power source, microcontroller, sensors, and communication modules. The chassis serves as the structural foundation, providing stability and support for other components. Motors, often DC motors or stepper motors, drive the wheels, enabling movement and navigation. Wheels play a crucial role in determining the robot's mobility and terrain traversal capabilities. Various wheel designs, such as omnidirectional wheels or differential drive systems 2.17, offer different advantages in terms of maneuverability and efficiency. The power source, commonly rechargeable batteries, provides the necessary energy for locomotion and onboard electronics.

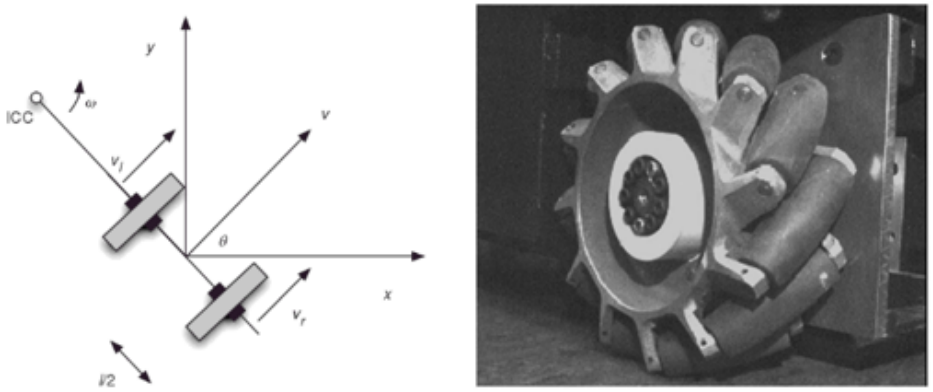


Figure 2.17: Differential Drive kinematics and mecanum omnidirectional wheel.Adapted from [16]

Shown in figure2.17, by varying the velocities of the two wheels, trajectories that the robot takes can be altered. As the rate of rotation ω about the ICC must remain consistent for both wheels, the following equations forming the basis of differential drive kinematics can be written:

$$\omega(R + \tfrac{l}{2}) = V_r$$
$$\omega(R - \tfrac{l}{2}) = V_l$$

Regarding omnidirectional wheels, the robot's trajectory can be adjusted by independently controlling the rotation speed and direction. For instance, to move forward, all motors propelling the robot can be instructed to rotate forward at corresponding speeds. The power source, commonly rechargeable batteries, provides the necessary energy for locomotion and onboard electronics. The need for small, low power computational devices to serve as

the brain for these robot kits arise, hence the wide use of microcontrollers, such as Arduino or Raspberry Pi, facilitating sensor integration, motor control, and decision-making algorithms. The table presented below 2.1 provides a comparison of the aforementioned microcontrollers.

Table 2.1: Core Specifications Comparison

Feature	**Raspberry Pi 3 B+**	**Arduino Uno Rev 3**	**ESP32 DevKit**
Microprocessor	BCM2837B0	ATmega328P	ESP32
Operating System	Linux-based	Bare Metal	FreeRTOS
Connectivity	Ethernet, Wi-Fi, Bluetooth	USB	Wi-Fi, Bluetooth
GPIO Pins	40	14 digital, 6 analog	36
USB Ports	4 x USB 2.0	1 x USB Type-B	1 x micro USB
Memory	1GB RAM	2KB SRAM, 32KB Flash	520KB SRAM, 4MB Flash
Voltage	5.1V 2.5A	6-20V	5V-12V

Sensors, including ultrasonic sensors, infrared sensors, or cameras, gather environmental data essential for navigation, obstacle avoidance, and localization. Communication modules enable wireless connectivity for remote control or data exchange with external devices, the choice for communications protocols depends on the embedded microcontroller's specification as shown in figure 2.1.

Using wireless communication technologies, these robot kits are then controlled from a distance. These systems find applications in areas such as search and rescue, surveillance, environmental monitoring, and space exploration. Advanced remote navigation systems incorporate features such as autonomous obstacle avoidance, real-time video streaming, and remote sensor data visualization, allowing operators to interact with the robot effectively in complex environments.

2.4 Discussion & Proposal

Section 2 discusses various algorithms enabling the computer vision task of object detection, which lays the foundation for many computer vision tasks such as object tracking and other user-specific applications. For this project, the YOLOv8 open-source model is adapted to augment the view of the robot. The augmentations include object detection and object tracking, which counts the number of objects of a certain class in view.

This class, of course, would be tailored to the user's specific needs at the time. The computer vision tasks being object detection and tracking would classify the system as superimposed AR system, which is a type of marker-less AR system, as discussed in chapter 2. The following entails the reasoning behind the YOLOv8 choice:

1. **Speed**
   YOLO models are designed for real-time object detection. They process images in a single pass through the network, which makes them significantly faster compared to multi-stage detectors like R-CNN variants. This speed is crucial for embedded systems with limited computational resources like Raspberry Pi.

2. **Efficiency**
   YOLOv8 is known for its efficiency in terms of both computational resources and memory usage. This efficiency is crucial for embedded systems, where resources are often constrained. YOLOv8 can achieve a good balance between accuracy and speed, making it suitable for deployment on devices like Raspberry Pi.

3. **Simplicity And Accuracy**
   YOLO models have a simpler architecture compared to multi-stage detectors like R-CNN variants. This simplicity makes them easier to implement and deploy on embedded systems, where complex models may be challenging to run efficiently. While traditional methods like HOG can be fast, they often lack the accuracy of deep learning-based models like YOLOv8. YOLOv8 can provide more accurate object detection results, especially for complex scenes and objects with varying scales and orientations.

4. **Adaptability**
   YOLOv8 can be fine-tuned on specific datasets or for specific tasks, making it adaptable to different applications. This flexibility is valuable for embedded systems where the requirements may vary from one deployment scenario to another.

Referencing Table 2.1, it is evident that the Raspberry Pi B+ emerges as the optimal selection for implementing computer vision algorithms among the available boards. Supported by its superior 1 GB RAM, extensive connectivity options, GPIO pins, and the Linux operating system, which implies broader support from a diverse user base, the Raspberry Pi is chosen as the central processing unit for the computer vision task. As elucidated in Chapter 5, the nature of the subsystems necessitates the incorporation of a secondary board tailored specifically for hardware processing, aligning with the architecture of the four-wheeled robot. Despite the Raspberry Pi offering considerable processing capabilities, the project's reliance on a remote navigation robot underscores the importance of mitigating

single points of failure within the system. Hence, the inclusion of two boards ensures continuity of functionality in the event of a failure in one part of the system. For instance, it remains critical to maintain visualization of the robot's environment in the event of a navigation system failure. The Arduino board is chosen as the cornerstone board for navigation, following are reasons behind the choice:

1. **Simplicity and Reliability**
   Arduino boards feature simpler architectures compared to Raspberry Pi and ESP32, rendering them more reliable in certain scenarios. For a relatively straightforward task like remote navigation, where the computational demands are modest, the simplicity of Arduino boards can prove advantageous. In this case, where resources must be allocated for two boards, opting for a simpler board is justified. Additionally, Arduino boards are less susceptible to crashes or freezes compared to more complex systems.

2. **Programming and Deployment**
   Arduino development is generally considered simpler due to its programming environment and extensive community support. It is arguably more straightforward to implement and deploy Bluetooth communication and motor control algorithms on an Arduino board compared to a Raspberry Pi or ESP32

Considering the inherent simplicity of mecanum omnidirectional wheels, the robot's architecture diverges from the differential drive kinematics depicted in Figure 2.17. Instead, it utilizes omnidirectional wheels for their simplified control in directional navigation tasks. The subsequent sections of this paper detail the methods employed to fulfill project requirements, along with the corresponding results, concluding with recommendations outlined in the plan of development. See figure 1.1

Chapter 3

Methodology

This section delineates the project life cycle, see figure 3.1, detailing the process followed, including revisions made to overcome obstacles encountered along the way. The study's life cycle comprises four phases: research, system design, implementation and testing, followed by apropriate revisions.

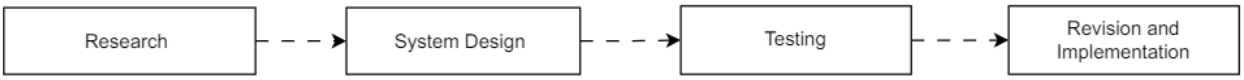


Figure 3.1: Project life cycle flow

3.1 Research

In Chapter 2, the research phase of the project life cycle is explored. This chapter delves into an investigation of relevant literature, derives fundamental design choices from the findings to meet project objectives . The contents of chapter 2 include the different open source computer vision algorithms, the different kinds of AR, robot architectures etc, these choices are further examined in Chapter 5, which focuses on the system design phase of the project.

3.2 System Design

In this phase, the system design is finalized, with the system divided into three subsystems: the *robot design and architecture*, *the computer vision*, and lastly, the *power subsystem* to provide sufficient power for all components. Each subsystem can be developed, tested, and maintained independently. This modularity simplifies the project management process and facilitates easier debugging and troubleshooting. This approach was adopted after considering that subsystems can be easily replaced or upgraded without affecting the entire project, enabling iterative development and continuous improvement over time. Moreover, given the time constraints of the project life cycle, this division not only aligns with best practices but also facilitates parallel progress across all subsystems, as thier development is independent of each other. For detailed insights into the design choices, refer to Chapter 5. Figure 3.2 shows the subsystems division and interaction, with the power module remaining as the joint system.

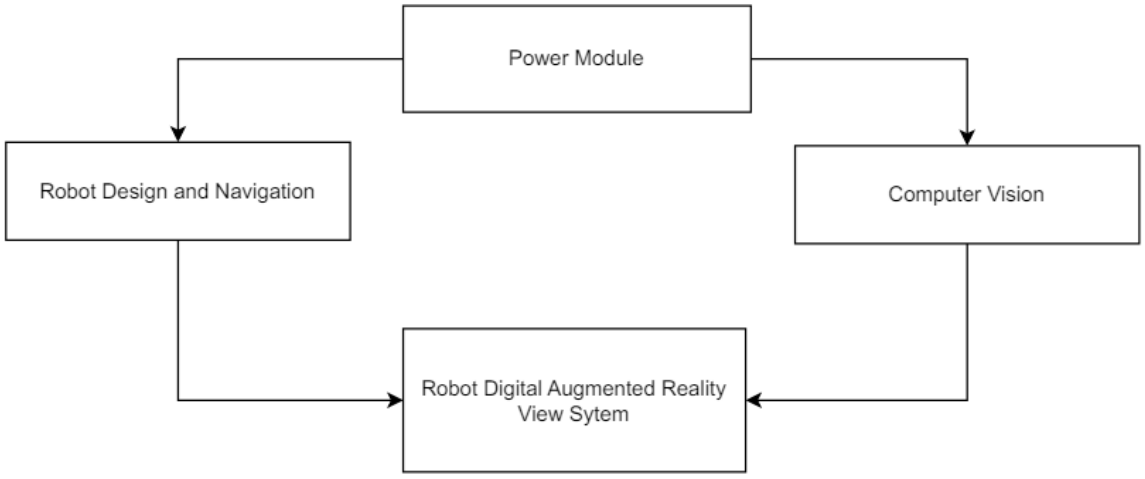


Figure 3.2: Interactions of the subsystems

3.2.1 Design choices

After partitioning the subsystems, design choices were made to align with project objectives and relevant literature, but mostly limited by the scope of the project. These decisions encompass selecting components for assembling the robot and ensuring sufficient power supply for all components, including both boards. Careful consideration is given to ensure seamless interfacing of all subsystems, guided by figure 3.2.

1. **Hardware Design**
   The system is required to operate as a standalone unit, thus it must host all components necessary for its functionality. Careful consideration has been given to the structure and size of the robot chassis, which will house all the components. The hardware design primarily focuses on the power module circuitry and the optimal placement of components to ensure optimal camera positioning for capturing feeds. The power module circuitry is designed to provide output to other subsystems with minimal wiring to reduce costs and complexity. Details of the system design are provided in section 5.

2. **Software Design**
   The system features software design that necessitates the computer vision tasks, object detection, and object tracking, which counts the number of instances of a selected class in the robot's view. Additionally, the design features a graphical user interface alongside the control unit to allow for the user to turn ON/OFF the augmentations from the control unit. Section 5 provides clear details on the approach taken to achieve the software objectives, based on Ultralytics YOLOv8 models as established in the research phase of the project life cycle. The model is trained on a custom dataset collected at the University of Cape Digital Image Processing Lab 1.

3.3 Implementation and Testing

In this phase of the project life cycle, the individual subsystems are implemented and tested. This process begins with testing the power module to ensure that all subsystems receive adequate voltage for proper functionality. After confirming the voltages, the robot's Bluetooth communication is tested by sending commands to evaluate its performance in terms of directional navigation, speed, range, and accuracy. Any necessary revisions are made to improve the system based on these tests. The custom-trained model is evaluated using the testing dataset, and its performance is analyzed. Due to the advanced capabilities of the YOLOv8 model and the scope of the project, extensive revisions for performance optimization are not required. This is also attributed to the chosen number of epochs during the algorithm's training. Further details are provided in the following system design section. The model is then deployed on unseen data, and conclusions are drawn based on its performance on this data. To consolidate the system, a graphical user interface (GUI) alongside the control unit is to be designed, with simplicity being a priority given the time constraints.

Chapter 4

System Design

The system design section aims to provide a detailed overview of the processes undertaken during the system's design life cycle. It is divided into hardware design and software design, each addressing the respective subsystems. Figure 4.1 provides system overview diagram used to as guidance to facilitate the system design.

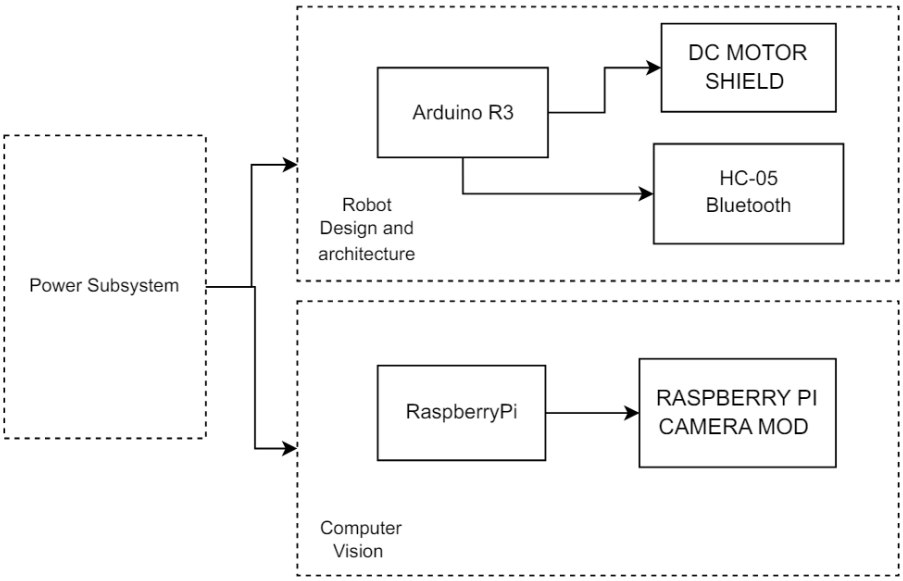


Figure 4.1: System design guide diagram

4.1 Hardware Design

The hardware design choices primarily focus on the robot chassis, which forms the structural framework of the robot and houses all the system components. The chassis was outsourced as part of the components for assembling the robot. Considering the system needs to accommodate an 85mm x 56mm Raspberry Pi 3 B+ board for computer vision tasks and a 68.6mm x 53.4mm Arduino Uno Rev3 for robot navigation, a 25.5cm long by 15cm wide chassis was selected. See Figure 4.2 below.

Figure 4.2: 4WD Aluminum Car Chassis With Mecanum Wheels

The chassis, made of aluminum, features two flat surfaces to house necessary components and has screws of universal sizes to ensure secure placement. This chassis was favoured due to its material as it potentially could minimize wiring costs and complexity, when used as ground. The following details outline the design choices and characteristics of each subsystem.

4.1.1 Robot design And Architecture subsystem

The design choices in terms of the robot architecture involved deciding on components that are relatively easy to interface to bring about a fully functional robot. section. below is a list of component choices chose:

1. **4WD MECANUM CHASSIS KIT-2**:2 Tier 60mm Aluminum Alloy Mecanum Wheel Chassis- 25.5cm Long X15cm Wide. With 4 X 3-6VDC TT Motors
2. **UNO REV3**:Arduino UNO Microcontroller Board Based on the ATMEGA16U2
3. **DC MOTOR SHIELD L293D 1.2A**
4. **BLUETOOTH MODULE HC-05 6PIN**

Figure 4.3 below shows the proposed pin connections with the chasis size in consideration.

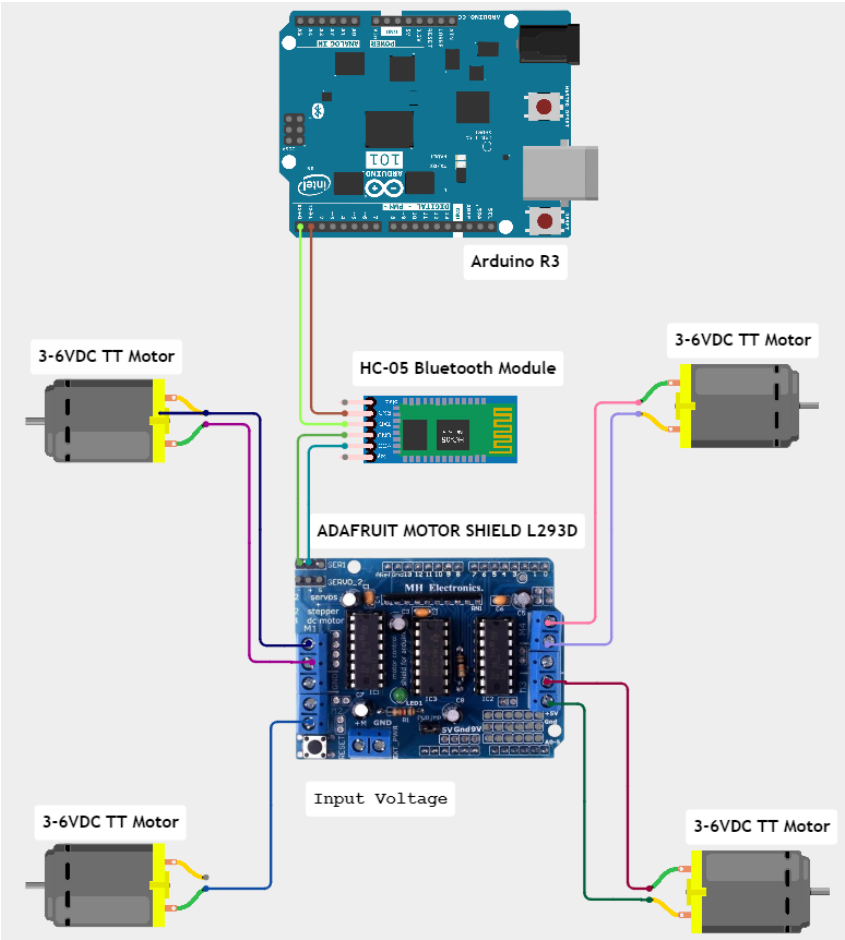


Figure 4.3: Robot design and components connection

The four Mecanum wheels are attached to four DC motors, which are connected to an Adafruit motor shield. This shield provides an interface between the Arduino boards and the DC motors, enabling precise and convenient motor control functionalities. The chosen L293D motor shield is advantageous compared to other motor shields because it mounts directly on top of the Arduino board, saving space and ensuring a stable connection that enhances the robot's stability. Additionally, the motor shield interfaces with the HC-05 Bluetooth module through serial communication.

The transmitter line of the HC-05 connects to the receiver pin of the Arduino, with pin R_x of the HC-05 soldered to shield pin 0 and pin T_x soldered to pin 1. These connections to R_x and T_x of the Arduino are established when the shield is attached on top of the board.

4.1.2 Power subsystem

The hardware design decisions determined the required output voltages from the power subsystem. The Raspberry Pi 3 B+ can be powered in three ways: via a 5V/2.5A DC micro USB connector, a 5V DC GPIO header, or through Power over Ethernet (PoE), which requires an additional PoE HAT. The Arduino Motor Shield must be powered by an external power supply due to the L298 IC on the shield, which has separate power connections for logic and motor supply. The motor current requirement often exceeds the USB current rating.

To address this, the Arduino is powered through the motor shield using its jumper functionality. When the jumper is connected, the Arduino receives input voltage from the shield and utilizes its voltage protection circuitry to prevent overvoltage.

The Arduino operates at 5V, while the HC-05 module requires 3.3V, which it draws from the shield, as depicted in Figure 4.3. The HC-05 includes a voltage regulator for VCC, allowing it to safely handle 5V. Consequently, the power module only needs to supply two voltage outputs: 5V for the Raspberry Pi and the L293D, which supports motor operation between 4.5VDC and 25VDC. The following circuit aims at addressing the power inputs.

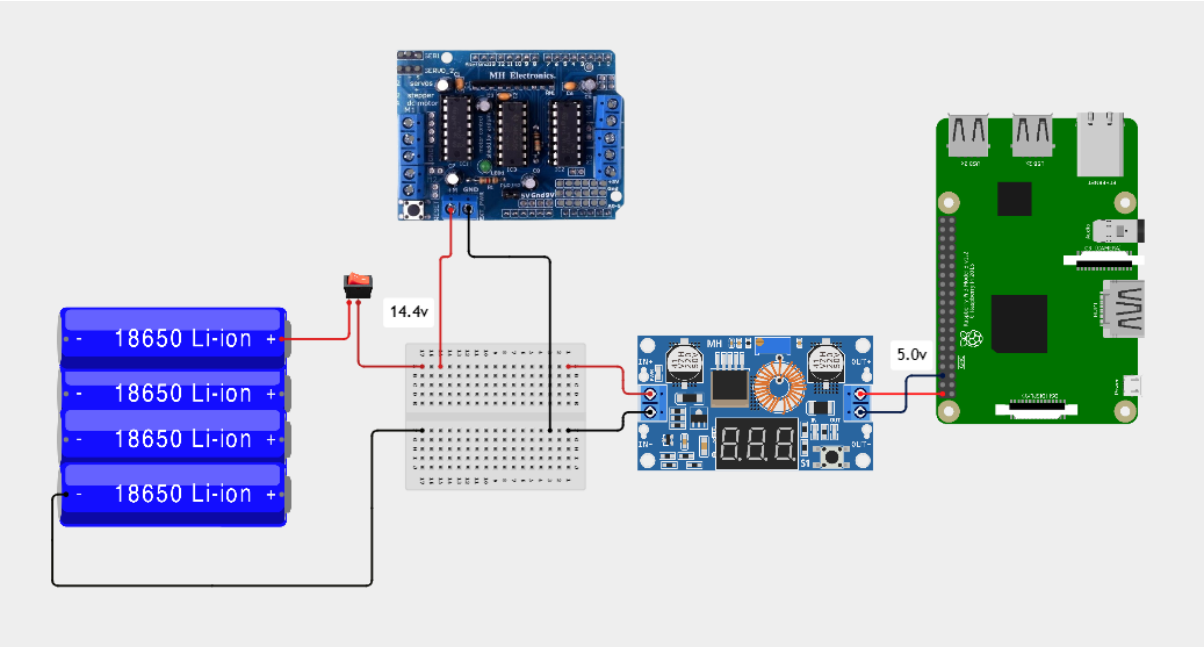


Figure 4.4: Power subsystem module

In Figure 4.4, four 3.6V rechargeable 18650 batteries are used as the voltage source. The total voltage of 14.4V is within the operating range of the L293D. A DC step-down

converter is used to reduce the supply voltage to 5V for the Raspberry Pi, providing sufficient voltage for all components. If the battery voltage drops below an unacceptable level, the motors will slow down, indicating the need for a recharge, and the Raspberry Pi's Linux system will also provide an indication. Additionally, when using remote desktop to connect to the Raspberry Pi, the signal will remain available, ensuring continued remote monitoring and control. The circuit also features a power button that serves as the main switch for the entire system. As shown in the figure, opening the switch disconnects the positive terminal from the battery, cutting power to all components. In implementation, the power system features a 4x18650 battery holder, and the DC buck step-down converter is soldered onto a veroboard along with header pins to connect jumpers to the respective components.

4.1.3 Computer Vision subsystem

The hardware design for the computer vision subsystem involved securely placing the camera at the front of the chassis to allow for an unobstructed view. Initial plans included 3D printing a structure to elevate the camera, attaching its base to the chassis. However, due to time constraints and a lack of available resources, the camera is attached to the chassis using a small rectangular piece of plank, which is also affixed to the chassis with double-sided tape, figures included in the appendix. The following section details the software designs aimed at creating the software component of the system.

4.2 Software Design

The software design involves the approach taken to achieve robot navigation for both the robot design and architecture subsystem and the computer vision subsystem. This includes programming the robot's direction, collecting data, and preparing to train the Yolov8 model for object detection, followed by object tracking to count the number of instances of a certain class in the object view. The nature of the power subsystem does not feature software, hence the exclusion of the subsystem in this section. Figure 4.5below presents an overview of the system software design.

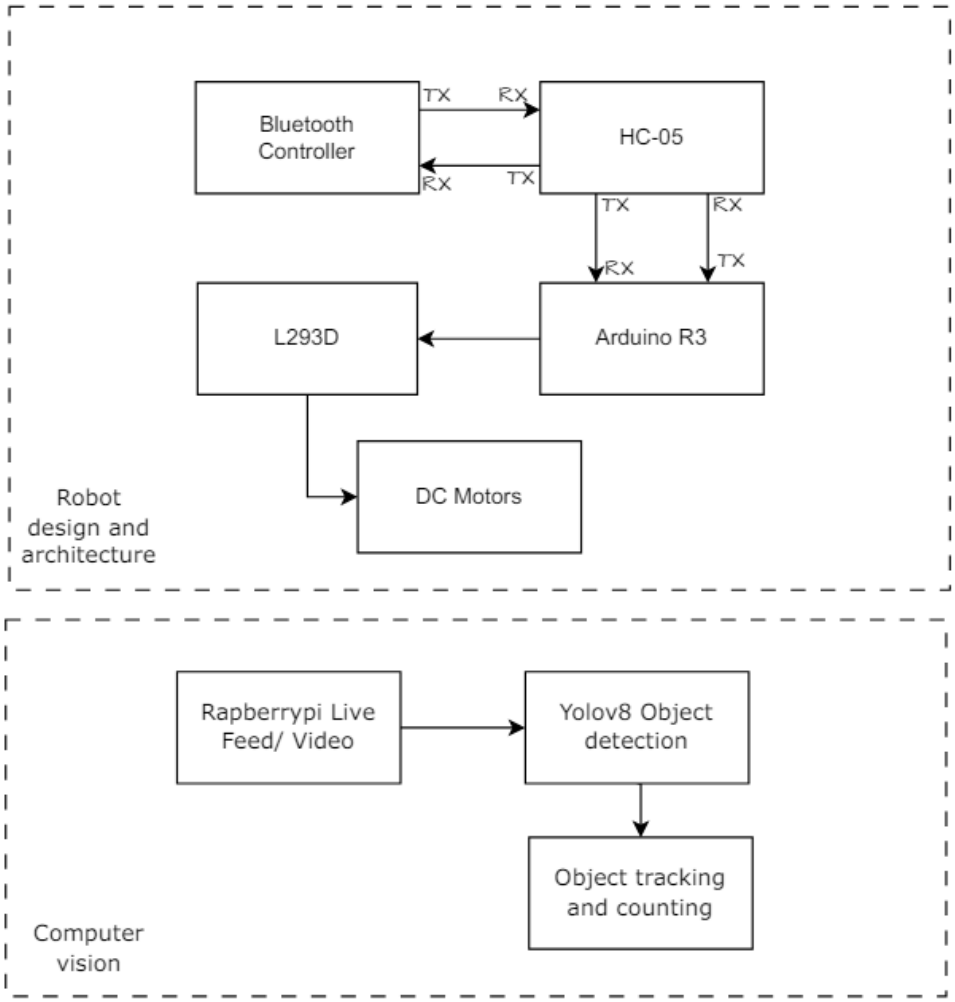


Figure 4.5: Software design overview

4.2.1 Robot design and architecture

In the reasearch phase of the project life cycle, the omnidirectional turning stands out as the prefered mechanism. Therefore the Bluetooth controller which was astablished to be a smart phone is designed to send commands to the HC-05 that trigger, forward, backwards, sideways(left and right) and clockwise direcitonal turning, see figure 4.6below.

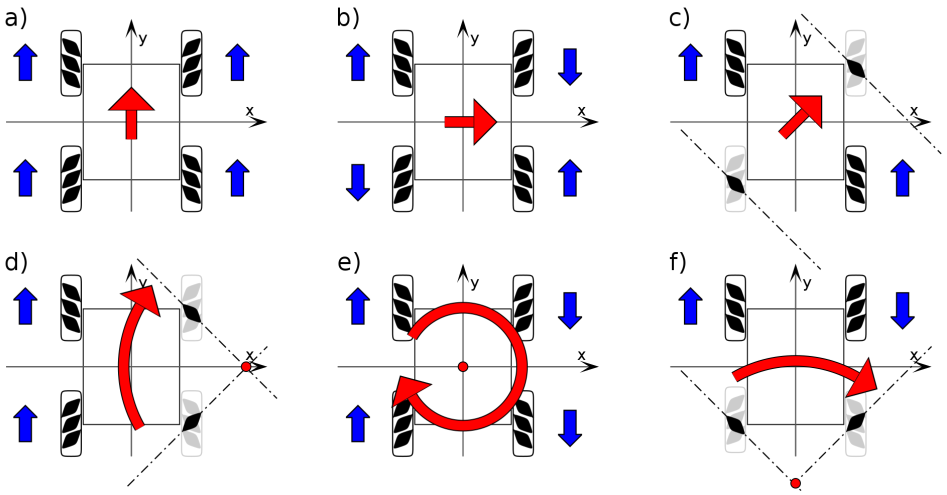
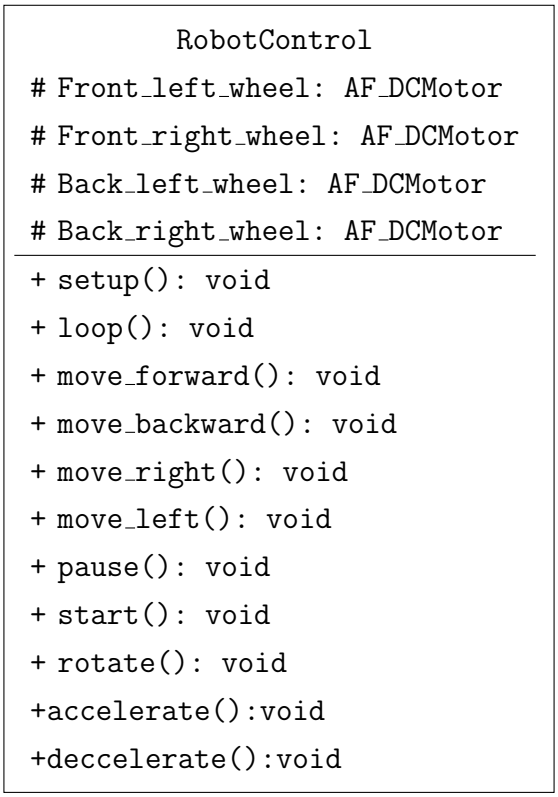


Figure 4.6: Mecanum wheels turning. a) Moving straight ahead, b) Moving sideways, c) Moving diagonally, d) Moving around a bend, e) Rotation, f) Rotation around the central point of one axle. adapted from [17]

For simplicity the robot only includes a, b and e shown in figure 4.6 as these are enough to achieve the desired turning. The core program responsible for navigation entails receiving commands from a smartphone utilizing a third-party application featuring a user-friendly graphical interface. These commands are then transmitted as characters to the HC-05 module, thereby activating various directional methods. The accompanying UML diagram furnishes a comprehensive overview of the code structure and the employed methods.



The provided UML class diagram illustrates the structure of the RobotControl class, which acts as the primary controller for the robot. It encompasses four AF_DCMotor objects representing the wheels, alongside attributes such as motor_speed and command. Notable methods include setup() for initialization, loop() for continuous command processing, and methods responsible for directional turning as depicted in figure 4.6. These directional methods encompass move_forward(), move_backward(), move_right(), move_left(), pause(), start(), rotate(), accelerate() and decelerate(), all adopting names discriptive of their functionality. Such organization facilitates seamless control and movement of the robot's motors, effectively responding to commands received from the HC-05 Bluetooth module. Refer to the Appendix for further insights into the code.

4.2.2 Computer vision

"The software design presented in this section, illustrated in Figure 4.5, encompasses programs dedicated to streaming live camera feeds from the Raspberry Pi to the object detection system.

The Raspberry Pi executes a Python script responsible for capturing video from the camera and transmitting it via a network socket. Simultaneously, a separate Python script runs on the host/laptop, establishing a connection with the Raspberry Pi's server to receive the video frames, which are subsequently displayed on the laptop screen. This configuration enables the utilization of the Raspberry Pi's camera as one of the webcams for the host, facilitating its integration as an input source for object detection. For additional details regarding the code, refer to the Appendix.

The following outlines the approaches employed for object detection and counting, to augment the view captured by the robot's raspberry pi camera.

Object detection

The section begins with the collection of the dataset as input to the YOLOv8 model. Data from the Digital Image Processing Lab at the University of Cape Town was gathered. Using the open-source RobotFlow platform, the data was labeled and prepared for training. The dataset consists of 26 classes, including rock, stop sign, and others (refer to the appendix), totaling 134 photos. The dataset is split into training, validation, and testing sets before being imported for the YOLOv8 training session. Given the small size of

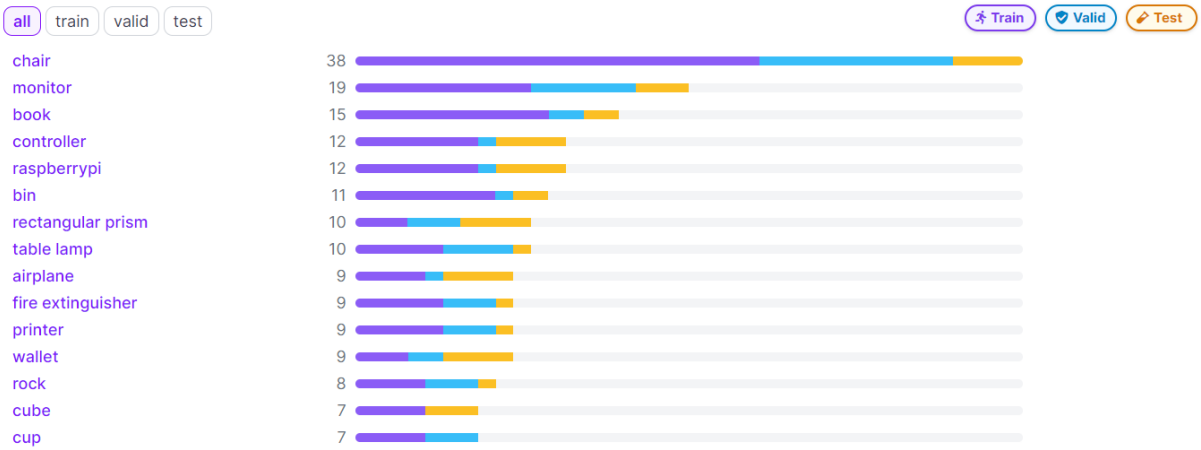


Figure 4.7: Snipet of the data split from Roboflow

the dataset, training epochs start at 10, increasing in intervals of 10. Performance metrics are monitored to determine the optimal stopping point for training. Additionally, cross-validation and data augmentation techniques were employed to populate more data, contributing to enhancing the model's robustness and accuracy.

Object tracking and counting

Upon training the model and obtaining satisfactory results, it was utilized in a class-counting algorithm. In the robot's view, all frames are processed with a line object optimally positioned to ensure that classes cross the line as the robot navigates closer to the object. Bounding boxes, class names, and incremented numbers are imposed on objects as they cross the line in the frame. This program draws inspiration from traffic monitoring systems that count vehicles on highways, but in this implementation, the robot moves toward the class objects. This approach has some limitations. Objects that have crossed the line in the initial view will not be counted, although they will remain in the robot's view. However, given the nature of the problem, which involves remote navigation, it is assumed that the robot will not initially be in an environment where it already has a view of the classes. The appendix contains the code responsible for this system. It should be noted that the training of the model and object counting were implemented on the Google Colab platform, utilizing the provided GPU for faster processing. The final implementation, however, is performed locally, subject to the host computer's limitations. The following sections presents the results obtained from method implementation after system design.

Chapter 5

Results

This chapter briefly presents the results obtained from the method implementation, testing the system designs. Figure 5.1 shows the fully connected system. More figures are included in the Appendix showing all system profiles.

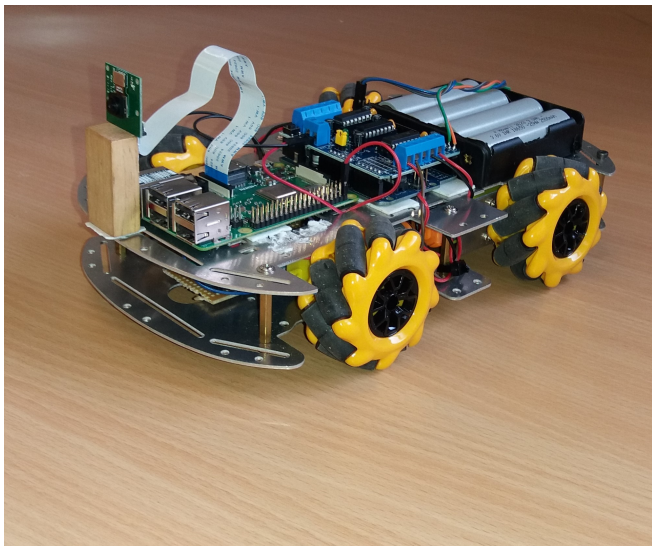


Figure 5.1: Fully connected camer-robot system.

5.1 Power Subsystem

The table below displays the voltages measured from the power subsystem upon connecting the system without any software implementations. For actual multimeter readings, please

refer to the appendix. These readings were essential to ensure that all system components have sufficient voltage for full functionality.

Table 5.1: Hardware Components and Voltage Readings

Component	Voltage Readings (V)
Back_right_wheel M1	0
Front_right_wheel M2	0
Front_left_wheel M3	0
Back_left_wheel M4	0
Raspberry Pi 3+B	5.1
L293D	14.23

5.2 Robot design And Architecture subsystem

Using the built-in Arduino IDE serial monitor and plotter, commands sent from the smartphone (used as the controller) are monitored. The required characters to trigger the directional turning, as shown in the code snippet (see Appendix A), were sent to the Arduino. The results from the serial plotter are presented below.

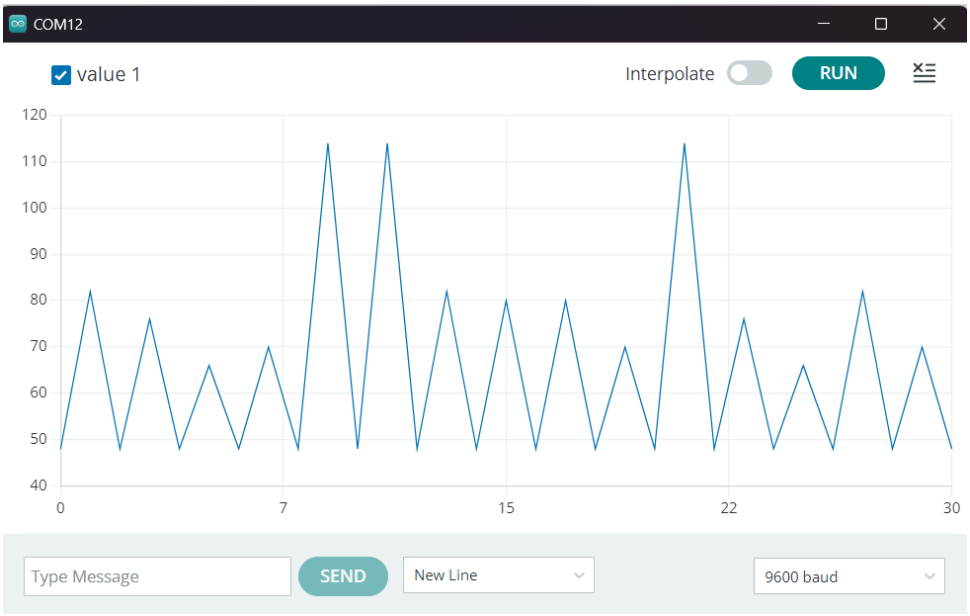


Figure 5.2: Arduino serial plotter

Figure 5.2 illustrates the characters received by the Arduino R_x via the HC-05 module. As depicted in Figure 9.1, the character 'F' is used to trigger the forward() function and is received as its ASCII value (70) at timestamps 7, 17, etc . This established that the

communication between the Arduino board and the cellphone was secured. Subsequent tests involved measuring voltages in the motor channels of the L293D upon the Arduino receiving commands. This was done to assess the voltage supplied to the motors, thereby testing both the power subsystem and the overall robot design and architecture. The table below presents the voltage results.

Table 5.2: L293D motor channels voltage readings

Motors	**Voltage Readings (V)**
Back_right_wheel M1	3.2
Front_right_wheel M2	3.1
Front_left_wheel M3	3.1
Back_left_wheel M4	2.90

The voltages confirmed the correct functionality of the navigation system; commands sent from the phone successfully triggered different directional turns. However, a slight time discrepancy was observed in the receipt of commands by the wheels, and at times, the motors did not receive equal voltages, as shown in Table 5.1. This resulted in negative implications: at higher speeds, the robot failed to follow directional commands accurately. Over time, the time discrepancy caused the robot to deviate from a straight path. However, this issue was less significant at lower speeds.

5.3 Computer vision

5.3.1 object detection

The YOLOv8 model was trained on a custom dataset, as shown in the code snippet below 5.3, for a varied number of epochs. The following presents the training results.

code > main.py > ...

```
1  ################################################################################
2  #Name: S'thabiso Lushaba
3  #Training YoloV8 object detection on a custom data set collected in menzies 6.16 Lab
4  #Student: LSHSTH002
5  ################################################################################
6
7  from ultralytics import YOLO
8
9
10 #building from scratch
11 model = YOLO("yolov8n.yaml")
12
13 results = model.train(data="data.yaml",epochs=10) #train the model
14
15
16
```

Figure 5.3: Custom Yolov8 training for 10 epochs.

60 epochs

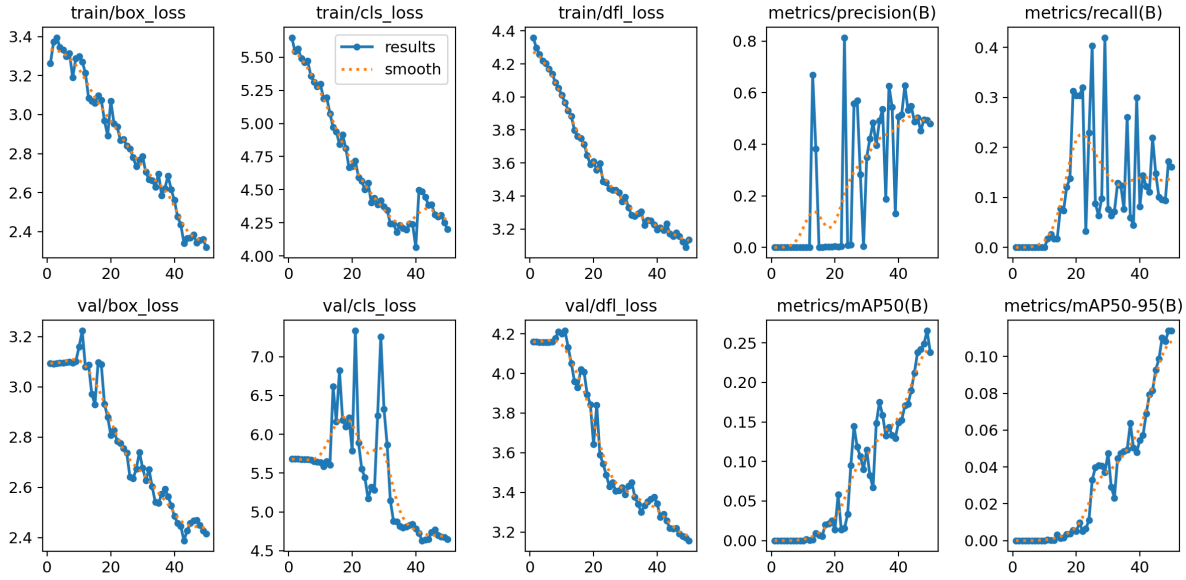


Figure 5.4: Training results; 60 epochs

300 epochs

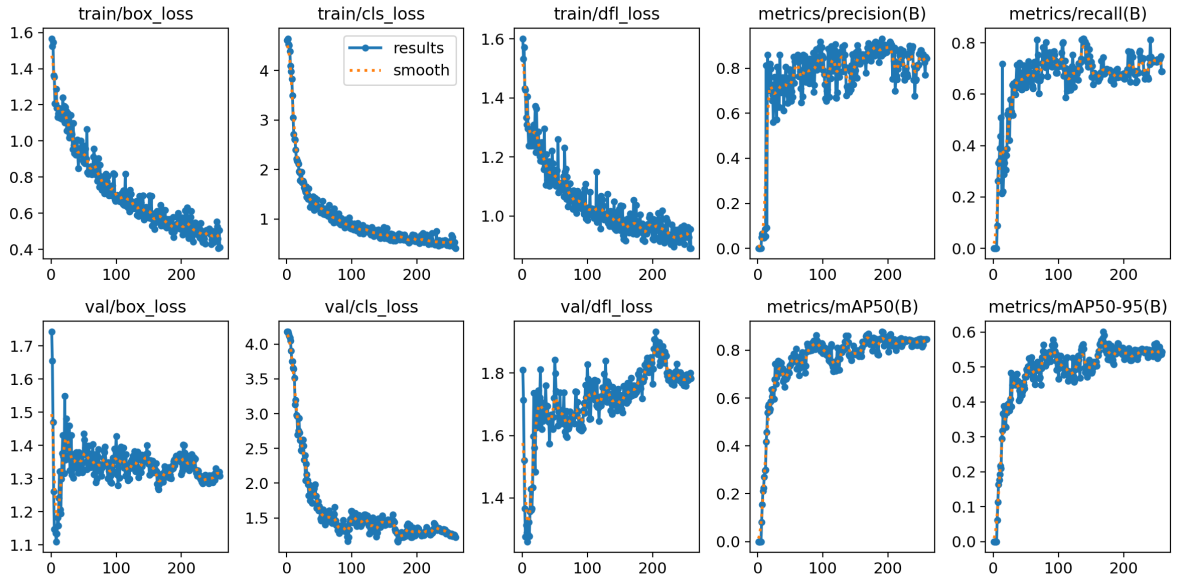


Figure 5.5: Training results; 300 epochs

The provided graphs depict various aspects of the training and validation performance, likely designed for an object detection task. A detailed explanation more focused on 5.5 as with more epochs that proved to be sufficient, see appendix a.

Training Losses

- **train/box_loss**: This graph represents the loss associated with the bounding box predictions. Over time, there is a steady decrease in this loss, indicating that the model is improving its ability to predict the precise locations of the objects' bounding boxes.
- **train/cls_loss**: This graph depicts the classification loss during training, which measures how well the model categorizes the detected objects into the correct classes. The decreasing trend signifies that the model's classification accuracy is improving over the training period.
- **train/dfl_loss**: The distribution focal loss, specific to the architecture of YoloV8 shows a decline showing that the model's performance, according to this specific metric, is improving as well.

Validation Losses

- **val/box_loss**: This graph shows the bounding box prediction loss on the validation data. The decrease followed by stabilization suggests that the model is effectively learning to predict bounding boxes on new data, indicating good generalization.
- **val/cls_loss**: This graph displays the classification loss on the validation set. A downward trend here signifies that the model is improving its classification performance on unseen data, similar to its training performance.
- **val/dfl_loss**: This graph represents the validation loss. The decrease followed by stabilization or a slight increase is normal and indicates that the model is generalizing well, though significant increases might suggest overfitting.

Performance Metrics

- **metrics/precision(B)**: Precision measures the accuracy of the positive predictions, the proportion of true positives among all predicted positives. The high and stable values in this graph indicate that the model is making accurate object detections without many false positives.
- **metrics/recall(B)**: Recall measures the ability of the model to detect all relevant objects, the proportion of true positives among all actual positives. The rapid

increase and subsequent stabilization show that the model is becoming adept at finding most of the objects in the images.

- **metrics/mAP50(B)**: Mean Average Precision at a 50% Intersection over Union (IoU) threshold is a common metric to evaluate object detection performance. The upward trend and high values suggest that the model is performing well in both detecting and correctly classifying objects at this IoU threshold.
- **metrics/mAP50-95(B)**: This metric measures the mean average precision averaged over multiple IoU thresholds (from 50% to 95%). The increase and stabilization of this metric indicate that the model performs robustly across various levels of detection strictness, signifying comprehensive detection performance.

Overall, these graphs collectively show a model that is learning effectively and improving over time. The decreasing trends in both training and validation losses indicate successful learning and good generalization to new data. The high and stable precision and recall metrics suggest that the model is making accurate and comprehensive detections. The mean average precision metrics further confirm the model's robustness in performance across different detection thresholds. To decide on the appropriate number of epochs, losses and metrics were monitored and upon stabilizing, it was deemed appropriate to conclude training as further improvements could be marginal.

5.3.2 Object tracking and counting

The object tracking and counting process initially utilized video footage depicting highway traffic due to its accessibility. The YOLOv8 model, instead of a custom-trained variant, was employed for this purpose. However, numerous challenges surfaced during implementation, primarily stemming from discrepancies in methods and attributes following updates by Ultralytics. These discrepancies led to potential dysfunctions between libraries. Ensuring seamless integration among all libraries proved to be a significant challenge, resulting in suboptimal implementation. The results obtained using the open-source YOLOv8 model and the traffic video feed are presented below. Subsequently, these results can be substituted with those obtained using the custom-trained model and the live video feed from the Raspberry Pi.

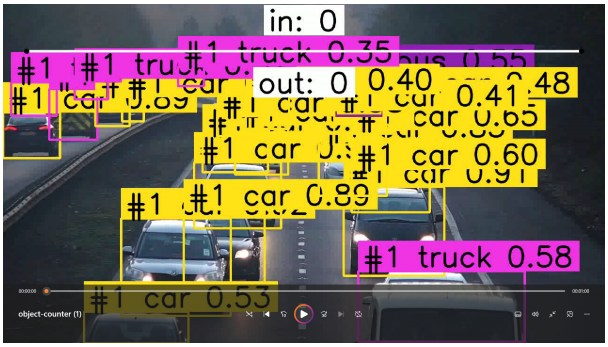


Figure 5.6: Object tracking and counting

In the depicted figure, the program initializes all detected instances as 1 at 0 seconds to disregard those preceding the line at the beginning of the counting process. This line is visible at the top of the frame and tallies the cars as they near the exit of the frame or highway. In practical application, this algorithm can be utilized for the remote navigation systems by adjusting the line's position to a desired location and transitioning to the custom-trained model. However, implementing this in the lab on the data the model was trained on was challenging as the space in confided and some classes are elevated in positions the line could not cover.

5.4 Implementation

This section presents the results of object detection implementation using the Raspberry Pi camera, elevated to enhance the likelihood of capturing objects in Digital Image Processing Lab 1. The detections are displayed only when the model's confidence level is at least 0.5, filtering out undesirable results.

Figure 5.7: Stop sign class detection

Figure 5.8: Tractor detections

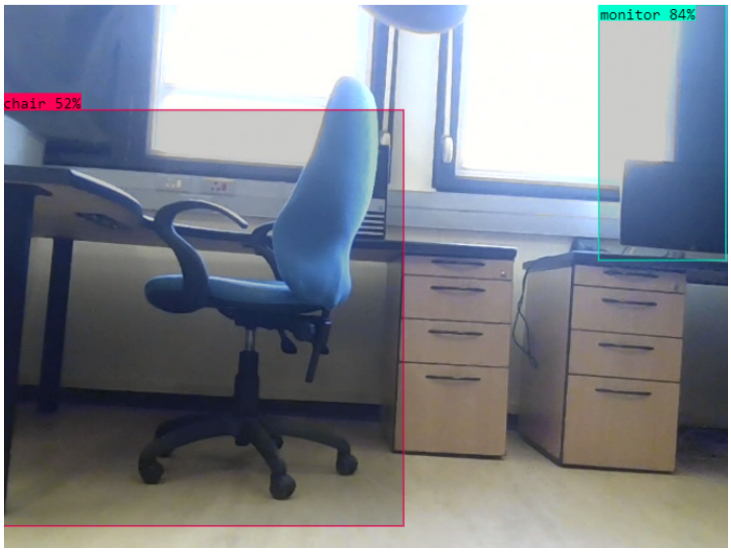


Figure 5.9: chair and monitor class detections

Figure 5.10: monitor class detections

The YOLOv8 model, trained for object detection, exhibited remarkable proficiency during implementation as can be seen in the above figures, reliably identifying target classes with confidence levels surpassing 50%. Integration of this model into the Raspberry Pi camera affixed to the robot facilitated an augmented reality view, enabling enhanced navigation and environmental interaction capabilities.

Chapter 6

Discussion

The results obtained from the object detection implementation using the YOLOv8 model underscore its significant proficiency in reliably identifying target classes with confidence levels surpassing 50%. This achievement is evident in the successful detection of a diverse range of objects, including stop signs, tractors, chairs, and monitors, as depicted in Figures 5.7 to 5.10. Such robust performance across various scenarios highlights the model's adaptability and effectiveness in real-world applications. Integration of the YOLOv8 model into the Raspberry Pi camera affixed to the robot represents a pivotal advancement, as it not only enhances the robot's navigation capabilities but also augments its overall perception of the surrounding environment. By leveraging the model's detection capabilities, the robot is empowered with an augmented reality view, enabling it to make informed decisions and navigate complex environments with greater accuracy and efficiency. Furthermore, the performance of the robot's power subsystem and overall architecture were meticulously evaluated to ensure seamless integration and functionality. Table 5.1 presents the voltage readings measured from the power subsystem, confirming adequate voltage levels for all system components. Additionally, the robot's design and architecture subsystem were assessed using the built-in Arduino IDE serial monitor and plotter. Results from the serial plotter revealed secure communication between the Arduino board and the cellphone, facilitating successful directional turns triggered by commands sent from the phone. However, slight discrepancies in the receipt of commands by the wheels and variations in motor voltages, as shown in Table 5.2, were observed. These discrepancies occasionally resulted in inaccuracies in directional commands, particularly at higher speeds, causing the robot to deviate from its intended path over time. Nevertheless, this issue was mitigated at lower speeds, demonstrating the robustness of the navigation system under varying conditions.

Chapter 7

Conclusions

In conclusion, this project signifies a significant stride in the ongoing effort to seamlessly integrate augmented reality into remote navigation systems. A versatile robot kit was constructed using Arduino, complemented by the utilization of Raspberry Pi and its camera module. This amalgamation of hardware components establishes a robust platform for real-time environmental perception and interaction, thereby laying the groundwork for advanced navigation capabilities.

The successful implementation of the object detection using the ultralytics YOLOv8 model , serves as a testament to the system's enhanced perceptual capabilities. The capability to recognize and classify various objects in real-time represents a crucial milestone in the evolution of autonomous systems. Notably, the object counting process presented challenges, particularly in scenarios involving highway traffic. While the object detection aspect performed admirably, accurately tallying objects in dynamic environments proved to be a nuanced task. Addressing these challenges necessitates further investigation and refinement of the counting algorithms to ensure accurate and reliable performance in real-world scenarios.

Looking ahead, the insights gleaned from this project will serve as valuable lessons for future endeavors in the field of remote navigation. By addressing the observed limitations and continuously refining the system's functionality, strides can be made towards achieving comprehensive and reliable performance in practical applications. In summary, this project exemplifies the transformative potential of augmented reality in revolutionizing remote navigation. Through diligent engineering efforts and a commitment to excellence, the groundwork has been laid for future innovations, paving the way for enhanced autonomy and intelligence in navigating complex environments.

Chapter 8

Recommendations

Moving forward, there are several recommendations to enhance the effectiveness and robustness of the augmented reality system for remote navigation. First and foremost, refinement of object counting algorithms is crucial. The challenges encountered in accurately tallying objects, particularly in dynamic environments like highway traffic, highlight the need for further refinement. Exploring advanced techniques in computer vision and machine learning can improve the accuracy and reliability of object counting, ensuring precise navigation and decision-making. Additionally, integrating sensor fusion techniques can significantly enhance the system's perception capabilities. By combining data from multiple sensors such as cameras, LiDAR, and radar, the system can gain a more comprehensive understanding of its surroundings. This integration of diverse sensor modalities can lead to improved navigation accuracy and robustness, especially in challenging environments.

Optimization for real-time performance is another key recommendation. Ensuring seamless real-time operation requires optimizing the system's algorithms and software architecture. This optimization involves efficient code design, leveraging hardware acceleration techniques, and minimizing latency in data processing pipelines. Prioritizing real-time performance enables the system to respond swiftly to changing environmental conditions, enhancing overall navigation effectiveness. Moreover, enhancing the user interface and interaction design can significantly improve the user experience. Designing user-friendly interfaces for controlling the robot, visualizing sensor data, and receiving feedback on navigation status is essential. Incorporating interactive elements such as augmented reality overlays can further enhance situational awareness and navigation precision, making the system more intuitive and user-friendly.

Conducting extensive validation and testing in real-world scenarios is critical to assessing the system's performance comprehensively. Field trials in various environments, including urban, rural, and indoor settings, can evaluate the system's robustness, reliability, and adaptability. Thorough real-world testing helps identify potential limitations and areas for improvement, informing future development efforts. Furthermore, due to time constraints, the implementation of a graphical user interface (GUI) to toggle the augmentation ON/OFF was not realized. For future recommendations, incorporating a user-friendly GUI to control augmented reality functionalities would enhance the system's usability and accessibility.

Lastly, adopting a continuous iterative development approach is essential for ongoing improvement. Soliciting feedback from users and stakeholders, identifying areas for enhancement, and iteratively refining the system through successive development cycles are vital. Embracing a culture of continuous improvement ensures that the system evolves to meet evolving user needs and technological advancements, ultimately enabling safer, more efficient, and more intuitive navigation in diverse environments.

Bibliography

[1] Z. Zou, K. Chen, Z. Shi, Y. Guo, and J. Ye, "Object detection in 20 years: A survey," *Proceedings of the IEEE*, vol. 111, no. 3, pp. 257–276, 2023.

[2] A. Singh, "Hog feature: Histogram of oriented gradients, extraction more." https://www.analyticsvidhya.com/blog/2019/09/feature-engineering-images-introduction-hog-feature-descriptor/, 2024. Accessed: 2024-04-20.

[3] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 1, pp. 886–893 vol. 1, 2005.

[4] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Region-based convolutional networks for accurate object detection and segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 1, pp. 142–158, 2016.

[5] R. Girshick, "Fast r-cnn," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[6] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," 2016.

[7] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

[8] M. Hussain, "Yolo-v1 to yolo-v8, the rise of yolo and its complementary nature toward digital manufacturing and industrial defect detection," *Machines*, vol. 11, no. 7, 2023.

[9] L. C. Viveiros, "Augmented reality and its aspects: a case study for heating systems," Master's thesis, Instituto Politecnico de Braganca (Portugal), 2019.

[10] K. MIZOKAMI, "U.s. army begins rolling out augmented reality for soldiers." https://www.popularmechanics.com/military/research/a41176138/us-army-augmented-reality-goggles-soldiers/, September 2022. Accessed: 2024-04-03.

[11] "Past meets present in hong kong's new, innovative real-time tourism app — south china morning post." https://www.scmp.com/presented/news/hong-kong/topics/hong-kong-tourism/article/3126463/past-meets-present-hong-kongs-new?campaign=3126463&module=perpetual_scroll_0&pgtype=article, 2024. Accessed: 2024-04-21.

[12] "How to take a snapshot in pokémon go — tom's guide." https://www.tomsguide.com/how-to/how-to-take-a-snapshot-in-pokemon-go, 2024. Accessed: 2024-04-22.

[13] M. Mine, J. Baar, A. Grundhöfer, D. Rose, and B. Yang, "Projection-based augmented reality in disney theme parks," *Computer*, vol. 45, pp. 32–40, 07 2012.

[14] S. Poghosyan, "Section 9. pedagogy learning-oriented augmented reality technology," 02 2019.

[15] "The goliath — newsletter archive — history tours." https://www.beachesofnormandy.com/articles/The_Goliath/?id=88cd77fe36#, 2024. Accessed: 2024-04-30.

[16] G. Dudek and M. Jenkin, *Computational Principles of Mobile Robotics*. Cambridge University Press, 2 ed., 2010.

[17] "Mecanum wheel - wikipedia." https://en.wikipedia.org/wiki/Mecanum_wheel, 2024. Accessed: 2024-05-19.

[18] L. S. Dalenogare, G. B. Benitez, N. F. Ayala, and A. G. Frank, "The expected contribution of industry 4.0 technologies for industrial performance," *International Journal of production economics*, vol. 204, pp. 383–394, 2018.

[19] "The rising role of augmented reality and virtual reality in industrial automation." https://www.automation.com/en-us/articles/february-2024/augmented-reality-virtual-reality-industrial, 2024. Accessed: 2024-05-06.

[20] M. Eckert, J. S. Volmerg, and C. M. Friedrich, "Augmented reality in medicine: Systematic and bibliographic review," *JMIR mHealth and uHealth*, vol. 7, p. e10967, apr 2019.

[21] "Augmented reality in education." https://www.apple.com/uk/education/docs/ar-in-edu-lesson-ideas.pdf, 2024. Accessed: 2024-05-06.

[22] F. Alvi, "Research areas in computer vision: Trends and challenges." https://opencv.org/blog/research-areas-in-computer-vision/, 2024. Accessed: 2024-04-06.

[23] R. K. McConnell, "Method of and apparatus for pattern recognition." https://patentimages.storage.googleapis.com/89/93/1e/b0b4c47e67a1ff/US4567610.pdf, Jan 1986. Accessed: 2024-04-10.

[24] D. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision*, vol. 60, pp. 91–, 11 2004.

[25] A. Mohan, P. CP, and T. Poggio, "Example-based object detection in images by components," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 23, pp. 349 – 361, 05 2001.

[26] B. Song, Y. Wang, and L.-P. Lou, "Ssd-based carton packaging quality defect detection system for the logistics supply chain," *Ecological Chemistry and Engineering S*, vol. 30, pp. 117–123, MAR 2023.

[27] S. Khaleghian, H. Ullah, T. Kræmer, N. Hughes, T. Eltoft, and A. Marinoni, "Sea ice classification of sar imagery based on convolution neural networks," *Remote Sensing*, vol. 13, p. 1734, 04 2021.

[28] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 32, no. 9, pp. 1627–1645, 2010.

[29] "Geforce gtx titan x — specifications — geforce." https://www.nvidia.com/en-us/geforce/graphics-cards/geforce-gtx-titan-x/specifications/, 2024. Accessed: 2024-04-21.

[30] A. Benali Amjoud and M. Amrouch, *Convolutional Neural Networks Backbones for Object Detection*, p. 282–289. Springer International Publishing, 2020.

[31] D. A. Team, "Deep learning ai video analytics products— deepvision ai." https://deepvisionai.in/index.html, 2024. Accessed: 2024-04-12.

[32] Y. Chen, Q. Wang, H. Chen, X. Song, H. Tang, and M. Tian, "An overview of augmented reality technology," *Journal of Physics: Conference Series*, vol. 1237, p. 022082, jun 2019.

[33] "6 technologies to thank the 1960s for — techradar." https://www.techradar.com/news/world-of-tech/6-technologies-to-thank-the-1960s-for-650980, 2024. Accessed: 2024-04-01.

[34] R. Klamma, "History of augmented reality — the open augmented reality teaching book - create and code augmented reality!." https://codereality.net/ar-for-eu-book/chapter/introduction/historyar/, 2024. Accessed: 2024-04-01.

[35] J. Skirnewskaja and T. D. Wilkinson, "Automotive holographic head-up displays," *Advanced materials*, vol. 34, no. 19, p. 2110463, 2022.

[36] A. Edwards-Stewart, T. Hoyt, and G. Reger, "Classifying different types of augmented reality technology," *Annual review of cybertherapy and telemedicine*, vol. 14, pp. 199–202, 2016.

[37] "Types of ar – digital promise." https://digitalpromise.org/initiative/360-story-lab/360-production-guide/investigate/augmented-reality/getting-started-with-ar/types-of-ar/, 2024. Accessed: 2024-04-05.

[38] "Pokémon go." https://pokemongolive.com/?hl=en, 2024. Accessed: 2024-04-05.

[39] "5 mind-blowing projection based augmented reality examples." https://provenreality.com/augmented-reality/projection-based-augmented-reality-examples/, 2024. Accessed: 2024-04-05.

[40] M. R. Mine, J. van Baar, A. Grundhofer, D. Rose, and B. Yang, "Projection-based augmented reality in disney theme parks," *Computer*, vol. 45, no. 7, pp. 32–40, 2012.

[41] R. Aggarwal and A. Singhal, "Augmented reality and its effect on our life," in *2019 9th International Conference on Cloud Computing, Data Science Engineering (Confluence)*, pp. 510–515, 2019.

[42] S. Schauer, J. Letellier, and J. Sieck, "Augmentation of printed content with web-based technologies," in *2021 11th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, vol. 1, pp. 252–256, 2021.

[43] "R.u.r. — robot, automation, science fiction — britannica." https://www.britannica.com/topic/RUR, 2024. Accessed: 2024-04-22.

[44] "The origin of the word 'robot'." https://www.sciencefriday.com/segments/the-origin-of-the-word-robot/, 2024. Accessed: 2024-04-22.

Chapter 9

Appendix A

sketch_mar25a.ino

```
/*
Author: Sthabiso Lushaba
Student Number: LSHSTH002

This code controls a robot's movement using an HC-05 Bluetooth module and four DC motors.
It defines functions to move the robot forward, backward, left, right, rotate, pause, and start.
*/

#include <AFMotor.h> // Include the AFMotor library

// Define motor objects for each motor
AF_DCMotor Front_left_wheel(3);
AF_DCMotor Front_right_wheel(2);
AF_DCMotor Back_left_wheel(4);
AF_DCMotor Back_right_wheel(1);

int motor_speed = 100; // Maximum speed
int command; // This holds the command read from the HC-05 Bluetooth module

void setup() {
  Serial.begin(9600);  // Set the baud rate for serial communication
  // Initialize any other necessary setup code here

  // Set the speed for each motor
  Front_left_wheel.setSpeed(motor_speed);
  Front_right_wheel.setSpeed(motor_speed);
  Back_left_wheel.setSpeed(motor_speed);
  Back_right_wheel.setSpeed(motor_speed);
}

void loop() {
  // Check if there is data available from the serial port
  while(Serial.available() > 0) {
    // Read the command from the serial port
    command = Serial.read(); // If there is data, store it in command;
    Serial.println(command); // Print this to the serial monitor just to double check

    // Determine the action based on the received command
    if(command == 'F') {
      move_forward();
    } else if(command == 'B') {
      move_backward();
    } else if(command == 'L') {
      move_left();
```

sketch_mar25a.ino

```
  Front_left_wheel.run(RELEASE);
  Front_right_wheel.run(RELEASE);
  Back_left_wheel.run(RELEASE);
  Back_right_wheel.run(RELEASE);
}

// Function to start all motors (same as pause)
void start() {
  pause(); // Same as pause
}

// Function to rotate the robot
void rotate() {
  // Set the speed for all motors
  Front_left_wheel.setSpeed(motor_speed);
  Front_right_wheel.setSpeed(motor_speed);
  Back_left_wheel.setSpeed(motor_speed);
  Back_right_wheel.setSpeed(motor_speed);

  // Run motors to rotate the robot
  Front_left_wheel.run(BACKWARD);
  Back_left_wheel.run(BACKWARD);
  Front_right_wheel.run(FORWARD);
  Back_right_wheel.run(FORWARD);
}

// Function to accelerate the motor speed
void accelerate() {
  motor_speed += 10; // Increment motor speed by 10
  // If motor speed exceeds 250, set it to 0
  if (motor_speed > 250) {
    motor_speed = 0;
  }
}

// Function to decrease the motor speed
void decelerate() {
  motor_speed -= 10; // Decrement motor speed by 10
  // If motor speed goes below 0, set it to 0
  if (motor_speed < 0) {
    motor_speed = 0;
  }
}

```

sketch_mar25a.ino

```
  Front_left_wheel.run(RELEASE);
  Front_right_wheel.run(RELEASE);
  Back_left_wheel.run(RELEASE);
  Back_right_wheel.run(RELEASE);
}

// Function to start all motors (same as pause)
void start() {
  pause(); // Same as pause
}

// Function to rotate the robot
void rotate() {
  // Set the speed for all motors
  Front_left_wheel.setSpeed(motor_speed);
  Front_right_wheel.setSpeed(motor_speed);
  Back_left_wheel.setSpeed(motor_speed);
  Back_right_wheel.setSpeed(motor_speed);

  // Run motors to rotate the robot
  Front_left_wheel.run(BACKWARD);
  Back_left_wheel.run(BACKWARD);
  Front_right_wheel.run(FORWARD);
  Back_right_wheel.run(FORWARD);
}

// Function to accelerate the motor speed
void accelerate() {
  motor_speed += 10; // Increment motor speed by 10
  // If motor speed exceeds 250, set it to 0
  if (motor_speed > 250) {
    motor_speed = 0;
  }
}

// Function to decrease the motor speed
void decelerate() {
  motor_speed -= 10; // Decrement motor speed by 10
  // If motor speed goes below 0, set it to 0
  if (motor_speed < 0) {
    motor_speed = 0;
  }
}
```

Figure 9.1: Robot design and architecture

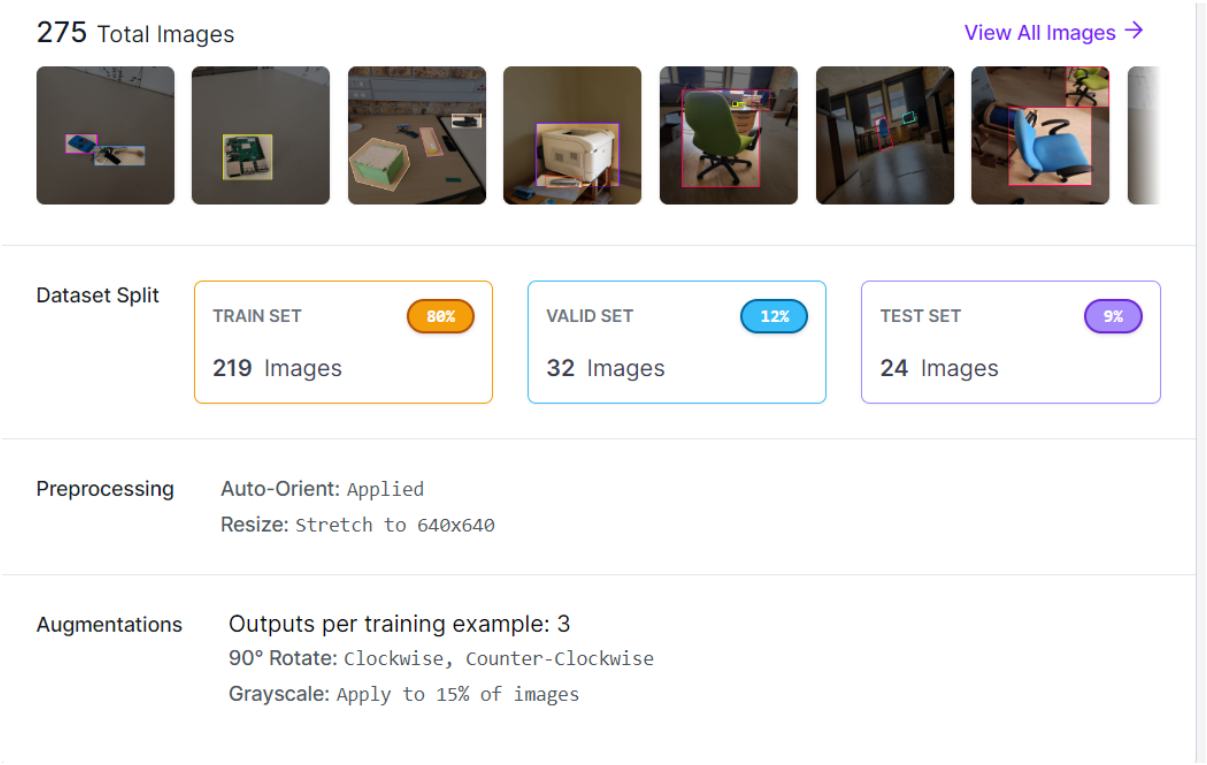


Figure 9.2: Dataset details

webcam.py > ...

```python
import cv2
import socket
import pickle
import struct

# Create socket
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client_socket.connect(('192.168.68.68', 9999))
connection = client_socket.makefile('rb')

try:
    while True:
        # Read frame length from the server
        data = connection.read(struct.calcsize("<L"))
        if not data:
            break
        # Unpack frame length and deserialize frame
        frame_size = struct.unpack("<L", data)[0]
        frame_data = connection.read(frame_size)
        frame = pickle.loads(frame_data)
        # Display frame
        cv2.imshow('Video', frame)
        cv2.waitKey(1)
except KeyboardInterrupt:
    connection.close()
    client_socket.close()
    cv2.destroyAllWindows()

```

Figure 9.3: Reading from pi camera as a webcam script

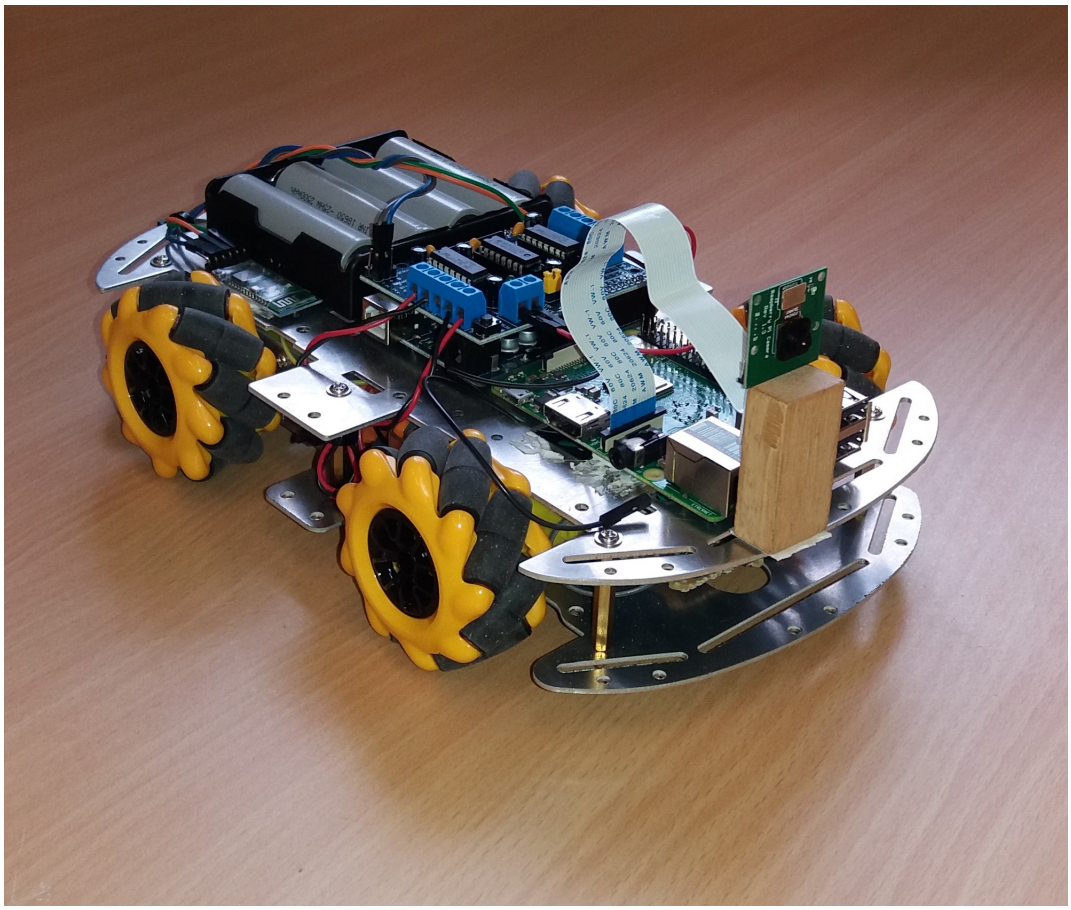


Figure 9.4: Fully connected robot view 1

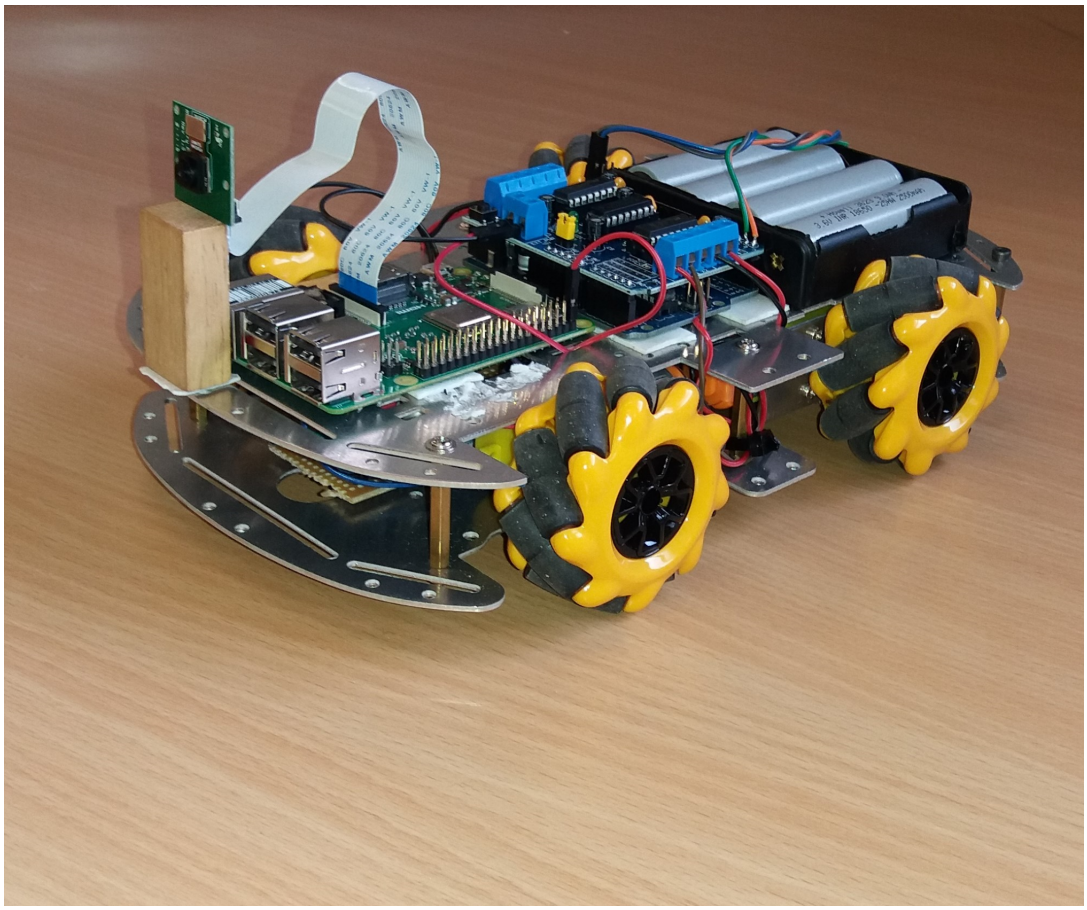


Figure 9.5: Fully connected robot view 2