DESIGN AND CONSTRUCTION OF A HEAD GESTURE CONTROLLED ROBOTIC CAR

BY

SANUSI OLAWUNMI SALIMAT

ENG1403490

EFAGENE VALENTINE EDESIRI

ENG1403447

DEPARTMENT OF COMPUTER ENGINEERING,

FACULTY OF ENGINEERING

UNIVERSITY OF BENIN

DECEMBER 2019

CERTIFICATION

This is to certify that this project work was carried out by SANUSI OLAWUNMI SALIMAT (ENG1403490) and EFAGENE VALENTINE EDESIRI (ENG1403447), students of the DEPARTMENT OF COMPUTER ENGINEERING, UNIVERSITY OF BENIN, in partial fulfilment of the requirements for the award of the BACHELOR of ENGINEERING DEGREE in COMPUTER ENGINEERING; under the supervision of ENGR. ISI A. EDEOGHON.

_______________________	_____________________
ENGR. ISI A. EDEOGHON	ENGR. DR. O. I. OMOIFO
PROJECT SUPERVISOR	HEAD OF DEPARTMENT
_______________________	_____________________
DATE	DATE

DEDICATION

This project is dedicated to Almighty God without whom this project work would not have been made possible and to our supervisor ENGR. ISI A. EDEOGHON.

ACKNOWLEDGEMENT

Our sincere appreciation goes to our project supervisor, ENGR. ISI A. EDEOGHON, for his insight and the guidance he gave us throughout the entire period of this project work.

Our heartfelt gratitude goes to our parent and siblings for their unrelenting support. We also acknowledge and appreciate the Head of Department and various lecturers who through lecturing us and its impact on us went a long way in understanding the implementation of this project.

ABSTRACT

The project involves building a robotic car and integrating with a camera. This robot is controlled by tracking a tag which is mounted on the forehead. Our objective is to control the robot using head gesture.

In order to achieve the aims and objectives, a gesture control system and car drive system has been designed to communicate wirelessly. The vision analysis chosen for the system is object tracking, this is used to detect the direction of head movement. The system then uses Wi-Fi socket communication to direct the low level mechanical parts (the car drive) after tracking.

Result shows accurate detection in a noisy By observing the results of testing the gesture control system is competent and it’s also enhance the natural way of intelligence by controlling the car movement with ease.

LIST OF FIGURES

LIST OF TABLES

CHAPTER ONE

INTRODUCTION

1.1 BACKGROUND OF STUDY

The need for help and control by humans has dated back to the very beginning of civilization, from the development of “Automata” in the 10th century to modern history of robots and computer evolution (Haug Walter, 1999). Our insatiable needs have made innovation continuous without a stop. This has brought technology to training computers to think and act like man, which is called artificial intelligence today. This has in no doubt helped living and sustainability of the earth.

Gesture has always been an important part of human communication. Gesture for man can been seen as shortcut for passing information.

In recent times man has sought to use gesture to automate and control things. A gesture controlled system enables a robot to act and think like human.

Life is filled with many colors and images therefore visual processing of things increases the accuracy and innovation of automation and technology as a whole. Computer scientists and engineers around the world have been trying to find ways to make machines extract meaning from visual data for about 60 years now, which is quite fascinating seeing the timeline of history.

In the1960s AI (Artificial Intelligence) became an academic discipline and some of the researchers were extremely optimistic about the its future, believed it would take no longer than 25 years to create a computer as intelligent as a human being (Dr. Chang Shu, 2008). This was the period when Seymour Papert, a professor at MIT’s AI lab, decided to launch the “*Summer Vision Project”* and solve, in a few months, the machine vision problem. He was of the opinion that a small group of MIT students had it in them to develop a significant part of a visual system in one summer. The students, coordinated by Seymour himself and Gerald Sussman, were to engineer a platform that could perform, automatically, background/foreground segmentation and extract non-overlapping objects from real-world images (Rostyslav Demush, 2009).

The project wasn’t a success. However, the project was the official birth of Computer Vision which began an evolution for computer systems and robotics (T.S Huang, 1996)

Computer Vision generally aims to give computers and machines visual understanding of the world.

1.2 PROBLEM STATEMENT

In recent years, there is an increasing need for control in research and everyday life, many tasks have gotten burdensome for humans and often times it is inconvenient or impossible for man to finish tasks due to environmental, health or general restrictions. With gesture control man will be able to perform task with little or no effort with effective and quick solutions to problems. This project uses head gesture (head movement) to drive a robotic car as a control prototype solving remote access control of devices, equipment as the case may be.

1.3 AIMS

The goal of this project is to design and implement a head gesture controlled s robotic car system.

1.3 OBJECTIVES

The following will help define the scope of the project and how it is intended to be implemented.

i. To develop a gesture control system using object tracking to capture and interpret head gesture.
ii. To develop a robotic car system.
iii. To design a user interface that displays the control data, navigation grid (object tracking result) and video field its window.

1.4 METHODOLOGY

The approach to this project is to basically implement remote wireless communication between device and the head gesture captured using computer vision analysis.

The chosen approach for vision analysis is comparison of color thresholds (color filtering) for object tracking. This solution provides generally accurate detection even in an environment which is noisy but has good color contrast.

Position and movement of object being tracked gotten from vision analysis of gesture is used to drive the robotic car via a wireless controller board (ESP8266) using Wi-Fi as medium of communication.

1.5 SCOPE OF STUDY

This study particularly focuses on remote or isolated control of automated devices using robotic car as a prototype. The design and implementation is specifically to control the movement of a robot using a gesture control system implemented with computer vision analysis.

Object tracking with color filtering system in computer vision is specifically adopted using HSV (Hue Saturation Value) color model which aligns closely with the way human-vison perceives color making attributes. The corresponding input (object movement) is used as direction navigation from this object tracking system is then used to drive a robotic car through a wireless communication with a controller.

1.6 RELEVANCE OF WORK

Computer vision control of devices using object tracking has a very wide range of application, some of which are discussed briefly below;

1. **Autonomous Navigation**: Autonomous driving uses computer vision control and this result to reduction of traffic accidents by eliminating human error, increasing road capacity and traffic flow by reducing distance between cars and making use of traffic management information, relieving the car occupants from driving and navigation activities and allowing them to engage in other activities or rest.

   Autonomous navigation finds its application in surveillance of infrastructures (pipelines, power lines, railways, waterways, roads, airports), search and rescue missions, mapping, fisheries, agriculture, forestry, natural resource monitoring, fire-fighting and emergency management, airborne communication collection and relay, weather data collection, environmental monitoring, pollution detection and other scientific research using computer vision.

2. **Gesture-based Human Computer Interface**: For example, using gesture (combined with speech) in interacting with virtual environments which is very efficient and productive for educational environments.
3. Wheelchair control for quadriplegics patients.

CHAPTER TWO

LITERATURE REVIEW

2.1 Gesture Control

Gesture control is the ability to recognize and interpret movements of the human body in order to interact with and control a computer system without direct physical contact. (Gartner, 2019)

A gesture is a form of non-verbal communication in which visible bodily actions communicate particular messages. It comprises of sound, light variation or any type of body movement. Based upon the type of gestures, they have been captured via Acoustic (sound), Tactile (touch), Optical (light), Motion Technologies, data glove, Bluetooth, infrared beams etc. Motion Technology has succeeded in drawing the attention of researchers from all over the world.

Gesture control is one of the growing technology in the world. Its application is endless;

- Handicapped people can control their wheelchairs and other gadgets using gestures.
- **It can bring a new edge to games. Today's mouse and keyboard-controlled games can be changed to gesture control. It would make games more realistic. (Instructables, 2017)**
- **Another application is in defense surveillance which is an important area of study. There is a continuous need for monitoring the land surveillance techniques in cases of natural disaster for search.**
-

2.2 Computer Vision

Computer vision is the science and technology that deals with how computers can be made to gain high-level understanding from digital images or videos.

Computer vision aims to build autonomous systems which could perform some of the tasks which the human visual system can perform (and even surpass it in many cases). Many vision tasks are related to the extraction of 3D and temporal information from time-varying 2D data such as obtained by one or more cameras. The properties and characteristics of the human visual system often give inspiration to engineers who are designing computer vision systems. Conversely, computer vision algorithms can offer insights into how the human visual system works.

2.3 OpenCV

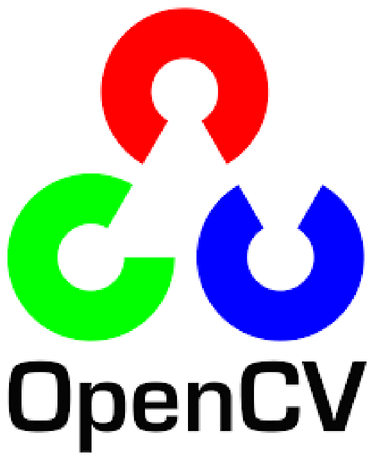


Figure 1 Open CV Logo

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being a BSD-licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

It is a collection of functions and classes that implement many Image Processing and Computer Vision algorithms. OpenCV is a multi-platform API written in ANSI C and C++, hence is able to run on both Linux and Windows.

The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc.

2.4 OpenFrameworks

Figure 2 OpenFramework

OpenFrameworks is an open source C++ toolkit designed to assist the creative process by providing a simple and intuitive framework for experimentation. OpenFrameworks is designed to work as a general purpose glue and wraps together several common used libraries, openCV library in this project.

2.5 Vision Analysis

2.5.1 Color Space

A range of colors can be created by the primary colors of pigment and these colors then define a specific color space. Color space, also known as the color model (or color system), is an abstract mathematical model which simply describes the range of colors as tuples of numbers, typically as 3 or 4 values or color components (e.g. RGB). Basically speaking, color space is an elaboration of the coordinate system and sub-space. Each color in the system is represented by a single dot.

A color space is a useful method for users to understand the color capabilities of the particular image to be tracked. It represents what the camera sees. There are a variety of color spaces, such as RGB, CMY, HSV, HIS. The RGB and HSV color space in this project.

2.5.1.1 RGB Color Space

RGB (R=Red, G=Green, B=Blue) is a kind of color space which uses red, green and blue to elaborate color model. An RGB color space can be simply interpreted as "all possible colors" which can be made from three colors for red, green and blue. In such conception, each pixel of an image is assigned a range of 0 to 255 intensity values of RGB components. That is to say, using only these three colors, there can be 16,777,216 colors on the screen by different mixing ratios.

2.5.1.1 HSB

(Hue, Saturation and Brightness)- is a model where three numbers represent each color. The first number, for hue, has a scale of 0 to 360 and includes all colors. The second number is for saturation. It runs from 0 to 100, with 0 equaling no color and 100 being full color. Brightness also runs from 0 to 100, with the higher number representing darker color.

2.5.2 Color Filtering

Filtering is a technique for modifying or enhancing the image that was captured. For example, you can filter an image to emphasize certain features or remove other features. Image processing operations implemented with filtering include smoothing, sharpening, and edge enhancement.

Filtering is a neighborhood operation, in which the value of any given pixel in the output image is determined by applying some algorithm in this project color algorithm to the values of the pixels in the neighborhood of the corresponding input pixel.

2.6 Arduino IDE

The Arduino Integrated Development Environment (IDE) is a cross-platform application (for Windows, macOS, Linux) that is written in functions from C and C++. It is used to write and upload programs to Arduino compatible boards, but also, with the help of 3rd party cores, other vendor development boards.

The source code for the IDE is released under the GNU General Public License, version 2. The Arduino IDE supports the languages C and C++ using special rules of code structuring. The Arduino IDE supplies a software library from the Wiring project, which provides many common input and output procedures. User-written code only requires two basic functions, for starting the sketch and the main program loop, that are compiled and linked with a program stub *main ()* into an executable cyclic executive program with the GNU toolchain, also included with the IDE distribution. The Arduino IDE employs the program *avrdude* to convert the executable code into a text file in hexadecimal encoding that is loaded into the Arduino board by a loader program in the board's firmware. The software can be used with any Arduino board and related modules and boards.

2.7 Microsoft Visual Studio IDE

Microsoft Visual Studio is an integrated development environment (IDE) from Microsoft. It is used to develop computer programs, as well as websites, web apps, web services and mobile apps. Visual Studio uses Microsoft software development platforms such as Windows API, Windows Forms, Windows Presentation Foundation, Windows Store and Microsoft Silverlight. It can produce both native code and managed code.

Visual Studio includes a code editor supporting IntelliSense (the code completion component) as well as code refactoring. The integrated debugger works both as a source-level debugger and a machine-level debugger. Other built-in tools include a code profiler, designer for building GUI applications, web designer, class designer, and database schema designer. It accepts plug-ins that enhance the functionality at almost every level, including adding support for source control systems and adding new toolsets like editors and visual designers for domain-specific languages or toolsets for other aspects of the software development lifecycle.

Visual Studio supports 36 different programming languages and allows the code editor and debugger to support (to varying degrees) nearly any programming language, provided a language-specific service exists. Built-in languages include C, C++, C++/CLI, Visual Basic .NET, C#, F#, JavaScript, TypeScript, XML, XSLT, HTML, and CSS. Support for other languages such as Python, Ruby, Node.js, and M among others is available via plug-ins. Java (and J#) were supported in the past.

The most basic edition of Visual Studio, the Community edition, is available free of charge. The slogan for Visual Studio Community edition is "Free, fully-featured IDE for students, open-source and individual developers". (Wikipedia, 2019).

2.8 C++

C++ is a general-purpose programming language created by Bjarne Stroustrup as an extension of the C programming language, or "C with Classes". The language has expanded significantly over time, and modern C++ has object-oriented, generic, and functional features in addition to facilities for low-level memory manipulation. It is almost always implemented as a compiled language, and many vendors provide C++compilers, including the Free Software Foundation, LLVM, Microsoft, Intel, Oracle, and IBM, so it is available on many platforms.

C++ was designed with a bias toward system programming and embedded, resource-constrained software and large systems, with performance, efficiency¸ and flexibility of use as its design highlights. C++ has also been found useful in many other contexts, with key strengths being software infrastructure and resource-constrained applications, (Stroustrup B. , 2014) including desktop applications, servers (e.g. e-commerce, Web search, or SQL servers), and performance-critical applications (e.g. telephone switches or space probes). (Stroustrup B. , 17 February, 2014)

C++ is standardized by the International Organization for Standardization (ISO), with the latest standard version ratified and published by ISO in December 2017 as *ISO/IEC 14882:2017* (informally known as C++17). The C++ programming language was initially standardized in 1998 as *ISO/IEC 14882:1998*, which was then amended by the C++03, C++11 and C++14 standards. The current C++17 standard supersedes these with new features and an enlarged standard library. Before the initial standardization in 1998, C++ was developed by Danish computer scientist Bjarne Stroustrup at Bell Labs since 1979 as an extension of the C language; he wanted an efficient and flexible language similar to C that also provided high-level features for program organization. C++20 is the next planned standard, keeping with the current trend of a new version every three years.

2.9 H-bridge

In general, an H-bridge is a rather simple circuit, containing four switching element, with the load at the center, in an H-like configuration:

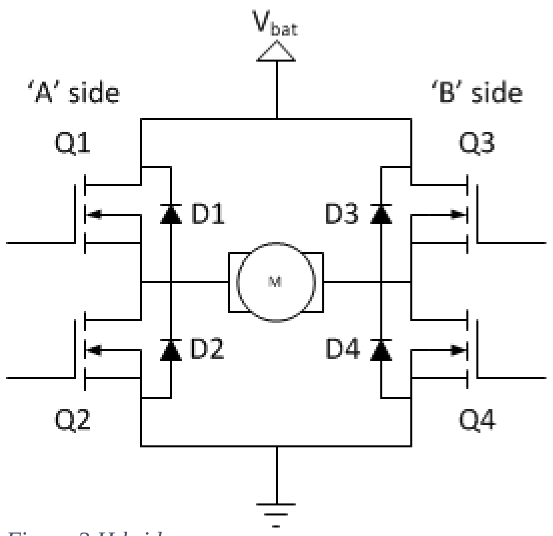


Figure 3 H-bridge a

The switching elements (Q1..Q4) are usually bi-polar or FET transistors, in some high-voltage applications IGBTs. Integrated solutions also exist but whether the switching elements are integrated with their control circuits or not is not relevant for the most part for this discussion. The diodes (D1..D4) are called catch diodes and are usually of a Schottky type.

The top-end of the bridge is connected to a power supply (battery for example) and the bottom-end is grounded.

In general, all four switching elements can be turned on and off independently, though there are some obvious restrictions.

Though the load can in theory, by far the most pervasive application if H-bridges is with a brushed DC or bipolar stepper motor (steppers need two H-bridges per motor) load.

2.10 COMPONENTS

2.10.1 ESP 8266

The **ESP8266** is a low-cost Wi-Fi microchip with full TCP/IP stack and microcontroller capability produced by manufacturer Espressif Systems in Shanghai, China. This microcontroller allows you to connect to a Wi-Fi network and make simple TCP/IP connections.

ESP8266 offers a complete and self-contained Wi-Fi networking solution, allowing it to either host the application or to offload all Wi-Fi networking functions from another application processor. When ESP8266 hosts the application, and when it is the only application processor in the device, it is able to boot up directly from an external flash. It has integrated cache to improve the performance of the system in such applications, and to minimize the memory requirements. Alternately, serving as a Wi-Fi adapter, wireless internet access can be added to any microcontroller-based design with simple connectivity through UART interface or the CPU AHB bridge interface.

ESP8266 on-board processing and storage capabilities allow it to be integrated with the sensors and other application specific devices through its GPIOs with minimal development up-front and minimal loading during runtime.

Figure 4 ESP8266 Board

2.10.1.1 Features of ESP 8266 Controller

- 802.11 b/g/n protocol
- Wi-Fi Direct (P2P), soft-AP
- Integrated TCP/IP protocol stack
- Integrated PLL, regulators, and power management units
- +19.5dBm output power in 802.11b mode
- Integrated temperature sensor
- Supports antenna diversity
- Power down leakage current of < 10uA
- Integrated low power 32-bit CPU could be used as application processor
- SDIO 2.0, SPI, UART
- STBC, 1×1 MIMO, 2×1 MIMO
- A-MPDU & A-MSDU aggregation & 0.4μs guard interval

- Wake up and transmit packets in < 2ms
- Standby power consumption of < 1.0mW (DTIM3)

2.10.1.2Architecture

General Purpose IO

There are up to 16 GPIO pins. They can be assigned to various functions by the firmware. Each GPIO can be configured with internal pull-up/down, input available for sampling by a software register, input triggering an edge or level CPU interrupt, input triggering a level wakeup interrupt, open-drain or push-pull output driver, or output source from a software register, or a sigma-delta PWM DAC. These pins are multiplexed with other functions such as host interface, UART, SI, Bluetooth coexistence, etc.

Digital IO Pads

The digital IO pads are bidirectional, non-inverting and tristate. It includes input and an output buffer with tristate control inputs. Besides this, for low power operations, the IO can also be set to hold. For instance, when we power down the chip, all output enable signals can be set to hold low.ESP8266 802.11bgn Smart Device Optional hold functionality can be built into the IO if requested. When the IO is not driven by the internal or external circuitry, the hold functionality can be used to hold the state to the last used state. All digital IO pins are protected from over-voltage with a snap-back circuit connected between the pad and ground. The snap back voltage is typically about 6V, and the holding voltage is 5.8V. This provides protection from over-voltages and ESD. The output devices are also protected from reversed voltage with diodes.

2.10.2 ESP 12E

ESP-12E Motor Shield is designed and developed by Shenzhen Doctors of Intelligence & Technology (SZDOIT). This large current motor driven module can compatible with ESP controller boards. This shield board is driven by the special excent large power full-bridge chip L293DD from the famous STMicroelectronics company, which can directly drive 2-channels DC motors or one-channel stepper motor. The driven current can be arrived at 1.2A. In this motor shield board, the IO port of ESP-12E Dev Kit is used as the control port. The logic chip configured inside can finish IC driven. Thus, the shield board has four ports: D1, D2, D3, and D3, which are used as PWMA (motor A), PWMB (motor B), DA (direction of motor A), and DB (direction of motor B), respectively.

In addition, this shield board has many pins, such as VIN, 3.3V, DIO, AIO, SDIO, UART, SPI, RST, and EN, thus can conveniently connect all kinds of sensors (e.g., temperature and humidity, buzzer, light, relay sensor, etc.).

The board is developed by the humanized design with a power switch, and thus user can control the on-off of power conveniently.

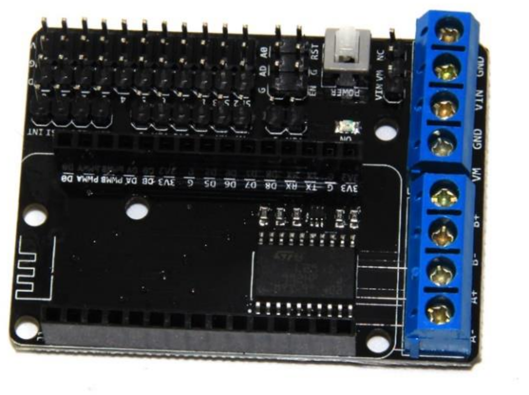


Figure 5 ESP 12E Motor Shield

Pin Assignment

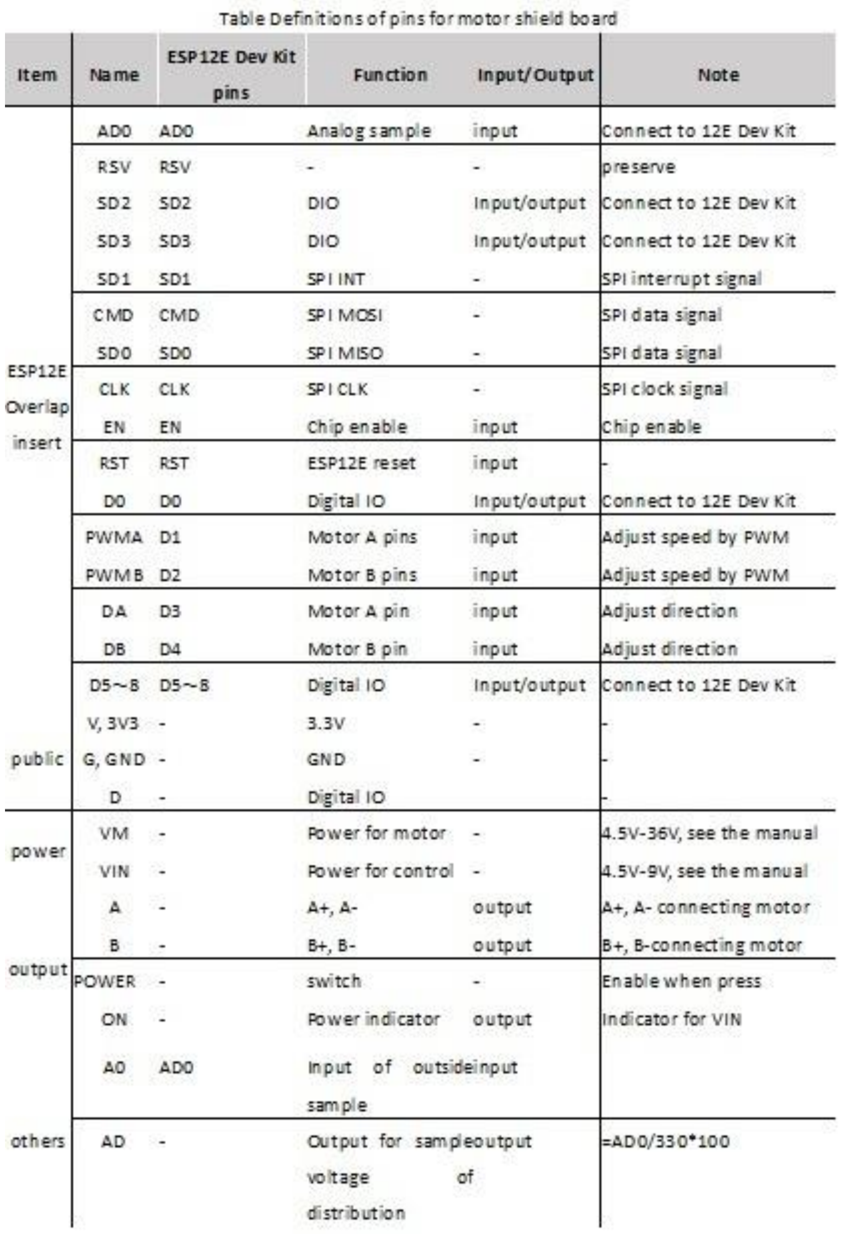


Table Definitions of pins for motor shield board

Item	Name	ESP12E Dev Kit pins	Function	Input/Output	Note
ESP12E Overlap insert	AD0	AD0	Analog sample	input	Connect to 12E Dev Kit
	RSV	RSV	-	-	preserve
	SD2	SD2	DIO	Input/output	Connect to 12E Dev Kit
	SD3	SD3	DIO	Input/output	Connect to 12E Dev Kit
	SD1	SD1	SPI INT	-	SPI interrupt signal
	CMD	CMD	SPI MOSI	-	SPI data signal
	SD0	SD0	SPI MISO	-	SPI data signal
	CLK	CLK	SPI CLK	-	SPI clock signal
	EN	EN	Chip enable	input	Chip enable
	RST	RST	ESP12E reset	input	-
	D0	D0	Digital IO	Input/output	Connect to 12E Dev Kit
	PWMA	D1	Motor A pins	input	Adjust speed by PWM
	PWMB	D2	Motor B pins	input	Adjust speed by PWM
	DA	D3	Motor A pin	input	Adjust direction
	DB	D4	Motor B pin	input	Adjust direction
public	D5~8	D5~8	Digital IO	Input/output	Connect to 12E Dev Kit
	V, 3V3	-	3.3V	-	-
	G, GND	-	GND	-	-
	D	-	Digital IO		-
power	VM	-	Power for motor	-	4.5V-36V, see the manual
	VIN	-	Power for control	-	4.5V-9V, see the manual
output	A	-	A+, A-	output	A+, A- connecting motor
	B	-	B+, B-	output	B+, B-connecting motor
	POWER	-	switch	-	Enable when press
	ON	-	Power indicator	output	Indicator for VIN
	A0	AD0	Input of outside sample	input	
others	AD	-	Output for sample voltage of distribution	output	=AD0/330*100

Figure 6 Pin Assignment of ESP 12E Motor Shield

2.9.2.1 Datasheet Specifications:

- Input Power: motor power (VM): 4.5~36V, can be powered separately; control power (VIN): 4.5V~9V (10V MAX), can be powered separately;
- Logic working current Iss:<=60mA (Vi=L), <=(Vi=H);

- Driven working current Io: <=1.2A;
- Max of dissipation power: 4W(T=90℃);
- Control signal input voltage: 2.3V<=VIH<=VIN (high), -0.3V<=VIL<=1
- Driven model: double ways large power H bridge driven;
- ESP-12E Dev Kit control port: D1, D3 (motor A); D2, D4 (motor B);
- Weight: about 20g.

2.9.3 ESP 32 Cam Board

The ESP32-CAM board is a $7 device that combines an ESP32-S chip and an OV2640 camera. It allows you to set up a video streaming web server, build a surveillance camera to integrate with your home automation system, do face recognition and detection, and much more. Besides the OV2640 camera, and several GPIOs to connect peripherals, the ESP32-CAM also features a micro SD card slot that can be useful to store images taken with the camera or to store files to serve to clients. (Santos, 2019).

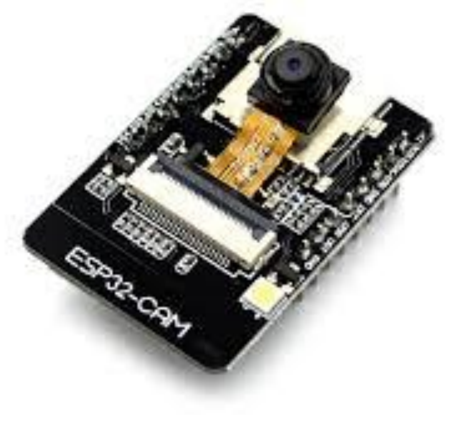


Figure 7 ESP32-CAM Board

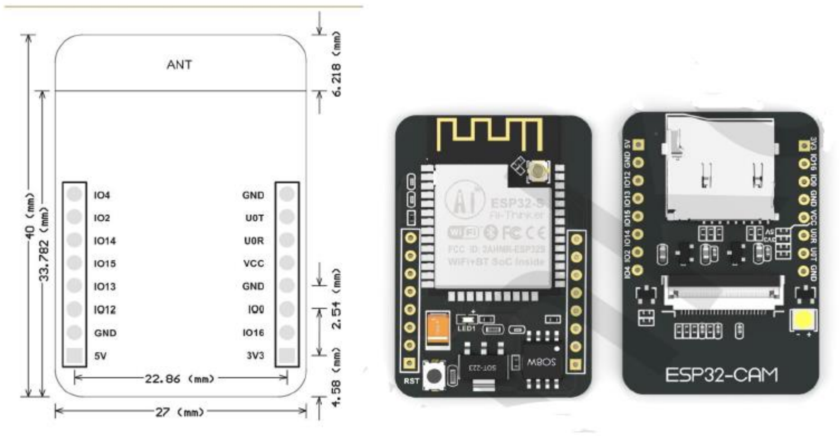


Figure 8 ESP32-CAM Pin Configuration

2.9.4 OV2460

The **OV2640** camera is a low voltage CMOS image sensor that provides the full functionality of a single-chip UXGA (1632x1232) camera and image processor in a small footprint package. The OV2640 provides full-frame, sub-sampled, scaled or windowed 8-bit/10-bit images in a wide range of formats, controlled through the Serial Camera Control Bus (SCCB) interface. This product has an image array capable of operating at up to 15 frames per second (fps) in UXGA resolution with complete user control over image quality, formatting and output data transfer.

Figure 9 OV2640 Camera

2.9.5 Arduino Uno

Arduino Uno is a microcontroller board based on the ATmega328P. It has 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs, a 16 MHz quartz crystal, a USB connection, a power jack, an ICSP header and a reset button. It contains everything needed to support the microcontroller; simply connect it to a computer with a USB cable or power it with a AC-to-DC adapter or battery to get started.

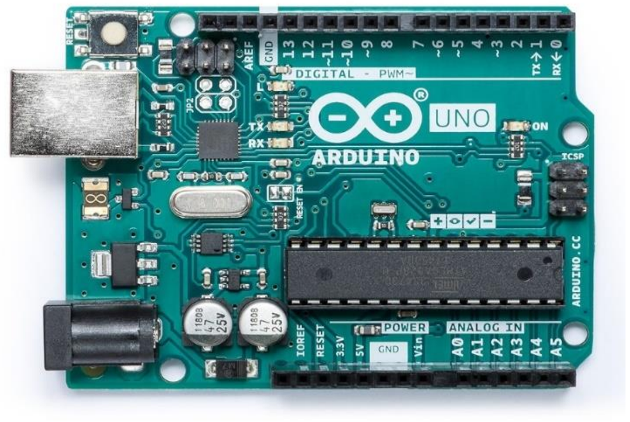


Figure 10 Arduino Uno Board

2.9.6 DC Motor

A **DC motor** is any of a class of rotary electrical machines that converts direct current electrical energy into mechanical energy. The most common types rely on the forces produced by magnetic fields. Nearly all types of DC motors have some internal mechanism, either electromechanical or electronic, to periodically change the direction of current flow in part of the motor.

DC motors were the first form of motor widely used, as they could be powered from existing direct-current lighting power distribution systems. A DC motor's speed can be controlled over a wide range, using either a variable supply voltage or by changing the strength of current in its field windings. Small DC motors are used in tools, toys, and appliances.

Figure 11 DC Motors

2.9.7 Connecting Wires

Connecting wires allows an electrical current to travel from one point on a circuit to another because electricity needs a medium through which it can move. Most of the connecting wires are made up of copper or aluminum.

Figure 12 Connecting Wires

2.10 RELATED WORKS

a. **Tianci Zhao and Peng Ren- "Intelligent Vision-Driven Robot for Sample Detection and Return"**: This project uses computer vision method to locate and return a user-specified object. It is implemented with a four-wheel mobile robot autonomously finds a path to the desired location using and efficient computer vision method. The robot uses it robust mechanical support that allows it travel through rough terrain. It was also implemented with accurate detection and a user-friendly interface program. Video Analysis using color filtering and haar cascading classification was used for autonomous tracking from its camera feed.
   <u>**Limitation**</u>- Wireless control between the system and the robot was not adopted and this decreased the detection range of the robot.

b. **Daniel J. Finnegan, 2012, Object Detection and Tracking in Images and Point Clouds**: This project was an attempt at developing an object detection and tracking system using modern computer vision technology. The project delivers an implemented tracking system.
   It consists of a hybrid of optical and modern infra-red technology and is applicable to areas such as unsupervised surveillance or semi-autonomous control. It is stable and is applicable as a stand-alone system or one that could easily be embedded into an even larger system.

Limitations- The tracker system cannot indicate the pose of any object it detects. The system is susceptible to a loss in detection if an object becomes occluded by another foreground object.

c. **Vladimir Kravtchenko, 1992, Tracking Objects in Real-time:** The goal of this research was efficient tracking of color objects from a sequence of live images for use in real-time applications including surveillance, video conferencing and robot navigation. Color representation of a dielectric object that models the behavior of a color cluster in color space that yields real time performance in segmenting out color object pixels. This representation accounts for non-white illumination, shadows, highlights, variable viewing and camera operating conditions. Then a clustering method that uses density and spatial cues to cluster object pixels into separate objects was used and a practical implementation of a tracking system based on the techniques was finally implemented.

Limitation- The camera movement was reactive to quick changes in the object dynamics

d. **Juha Kela, Panu Korpipaa, Sanna Kallio, 2014, Gesture Control System:**

Limitation:

e. Saurabh Kandalkar, Pratik Kadam, May 2015, Arduino Based Head Gesture Controlled Robot Using Wireless Communication: This project is about the robustness of Arduino based head movement controlled robot. This robot is controlled using motion sensor which is mounted on the head. In future there is need of robots which can be used to ease the human tasks and interact with the human easily. The objective is to control the robot using head gesture. Accelerometer is used to detect the direction of head movement. In order to full-fill the requirement a program was written and executed using a microcontroller system. By observing the results of experimentation the gesture formula is very competent and it's also enhance the natural way of intelligence and also assembled in a simple hardware circuit.

CHAPTER THREE

METHODOLOGY

This project design involves capturing a human gesture (head movement in this case) This gesture is interpreted and processed using computer vision to analyze its movements and positions. Data gotten from vision analysis is used to drive the robotic car via a wireless controller (ESP8266) using Wi- Fi as medium of communication. The car is to move/navigate according to the position of the head movement being captured. This chapter contains the detailed design approach and computational analysis adopted in the design and development of this project. The whole project is divided into two major units:

1. Gesture Control Unit
2. Robotic Car System Unit

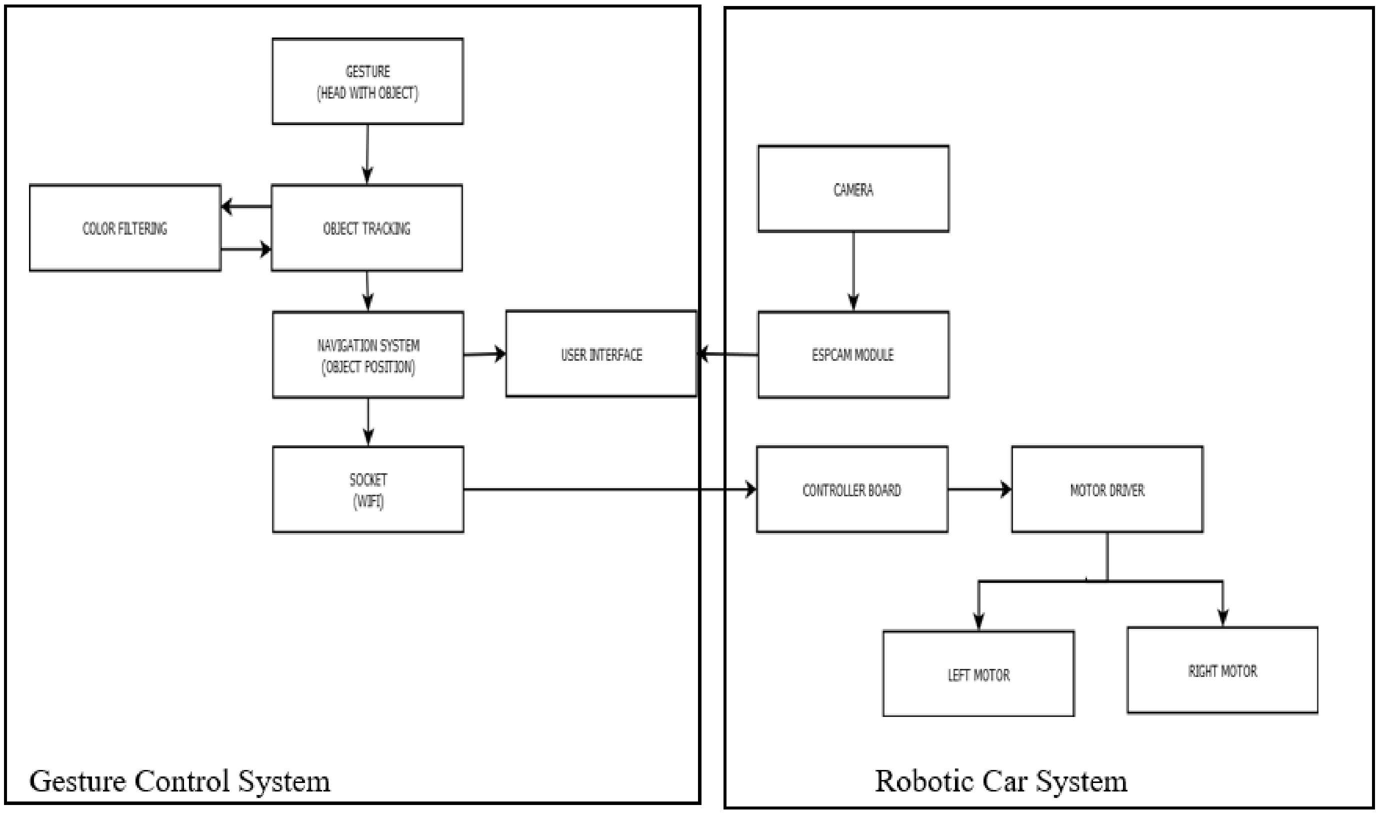


Figure 13 System Architecture Block Diagram

3.1 GESTURE CONTROL SYSTEM

Gesture control has to deal with the movement of the human body and the specific body part movement used for control in this project is “the human head”. This is interpreted and communicated via computer vision.

Computer Vision has many approach to it but vision analysis chosen is comparison of color thresholds (color filtering) for object tracking. This solution provides generally accurate detection even in an environment which is noisy but has good color contrast. This algorithm will be implemented using the openCV and openframeworks.

3.1.1 Gesture Capture (Head Movement)

This is the initial and starting point of the gesture control system. The gesture (forehead motion) is captured with the laptop webcam. Generally, in digital image processing, this is the first stage of any vision processing system which is called ‘image acquisition’.

Image acquisition in digital image processing is the action of retrieving image from a source (a webcam of a laptop) for processing. After the image has been obtained from the video, other required methods of processing are applied to it.

3.1.2 Object Tracking

Object tracking is an important task in many application areas like automated surveillance, human computer interfacing, vehicle navigation, traffic control, etc. There are three key steps in visual analysis: detection of interesting moving objects, tracking of such objects from frame to frame, and analysis of object tracks to recognize their behavior. Tracking can be defined as the problem of estimating the trajectory of an object in the image plane as it moves around a scene. The particular object to be tracked is a red tag attached to the forehead of the human.

3.1.3 Color Filtering Algorithm

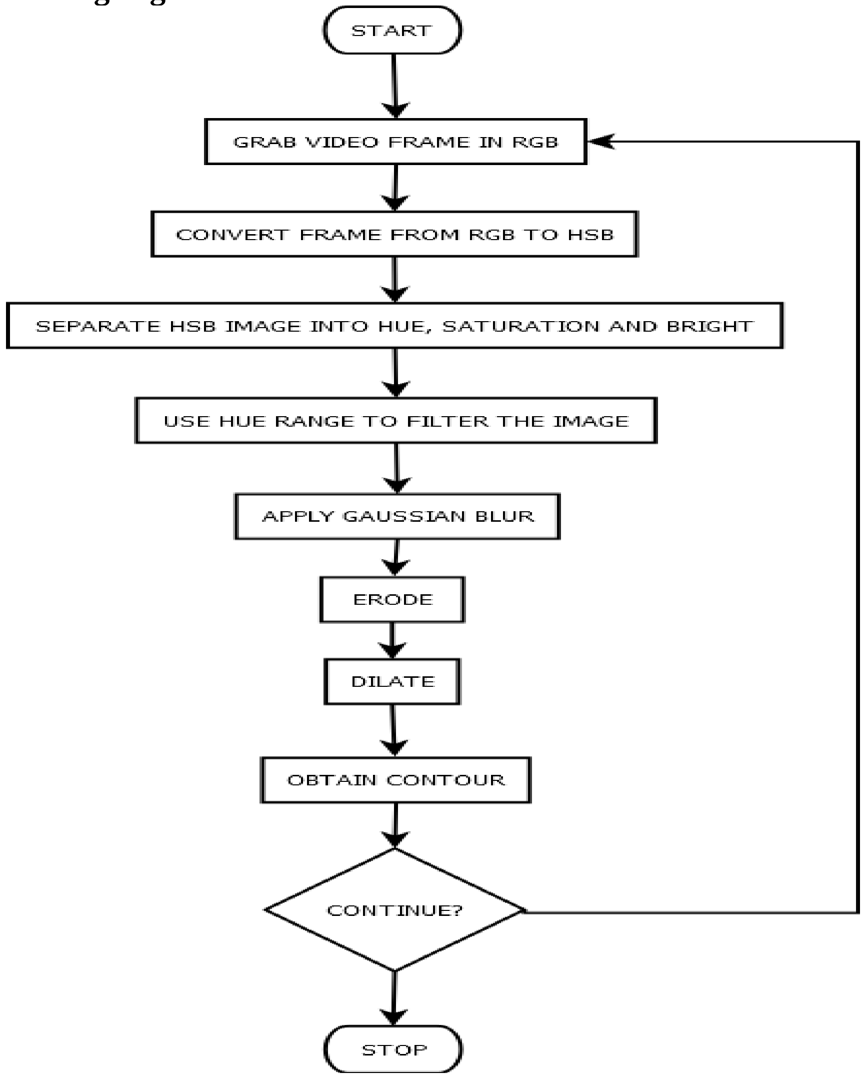

START
GRAB VIDEO FRAME IN RGB
CONVERT FRAME FROM RGB TO HSB
SEPARATE HSB IMAGE INTO HUE, SATURATION AND BRIGHT
USE HUE RANGE TO FILTER THE IMAGE
APPLY GAUSSIAN BLUR
ERODE
DILATE
OBTAIN CONTOUR
CONTINUE?
STOP

Figure 14 Color Filtering Flowchart

The choice for the vision algorithm is the color filtering/tracking method. This algorithm utilizes the fact the target object has a different color from the other objects in the environment to quickly locate the position of the targeted object. In real world experiments, we discovered that unless the object is put into an environment with many same color distractions, the algorithm works pretty well in detecting the object on the human forehead. Also, the accuracy and the success rate of the method is much better than other algorithms.

The color based object tracking method first assumes that there is a significant difference between the color of the target and the color of the environment. If the color contrast is not large enough, the performance of this method will be largely limited. As to the algorithm itself, the program first takes the captured image from the gesture capture as the source image. All vision analyzing and image processing be will based on this image. When the image has been captured from the camera, the program will blur the image to reduce the color noise. As the image is being blurred, some small areas of the distracting colors will be removed from the image; therefore the accuracy of the detecting will be increased.

OpenCV by default captures an image from the camera in a format of 8-bit, unsigned and RGB format. In other words, OpenCV captures an image consisted of 3 matrices—the red matrix, the green matrix and the blue matrix, each matrix element with integer values ranging from 0 to 255. As to the image, it consists of many small boxes, i.e. pixels that are too small to be distinguished by human eyes. However, since our algorithm does the color segmentation and isolation, the program needs to achieve a color space that is easy to do the separation. In fact, HSB color space is actually the most suitable color space for the color based isolation and segmentation. For this reason, the program will therefore convert the source image from RGB image to HSB image to convert the color space from the RGB color space to the HSB color space. As to the HSV image, it still consists of three matrices: the hue matrix, saturation matrix and brightness matrix. The ranges of the elements of the three matrices are 0 to 180, 0 to 255, and 0 to 255 respectively. To detect a specified color (red is selected as our color), the program must know the range of the hue, saturation and value of the target color to separate the color from the image. For instance, to detect a yellow object, the hue range for yellow is usually set to 20 – 40. Even though we could easily figure out the hue range for a specific color, the saturation and value of the color usually rely on the object materials and environment lighting; therefore, the specified saturation and value of the color must be determined according to the environment. After converting the image to an HSB image, the program will limit all the colors that do not match the color criteria according to the parameter ranges given by the user. Then the program will check if the areas detected are large enough be our target object.

3.1.3.1 Algorithm

1. Grab video: Obtain video feed from Computer's webcam.
2. Convert from RGB to HSB color space: Color tracking is notably better in HSB color space than in RGB.
3. Separate HSB into hue, saturation and brightness: This splits an image into its color planes, and writes them to three separate new images.
4. Get a filtered resultant image from the extracted hue image using a hue range: Set a range, and only colors within that range are retained.
5. Apply Gaussian blur: In this approach, a Gaussian kernel is used, which consists of values obtained from a Gaussian function. Gaussian filtering is highly effective in removing Gaussian noise from the image.
6. Erode: The basic idea of erosion is just like soil erosion only; it erodes away the boundaries of foreground object (Always try to keep foreground in white). So what does it do? The kernel slides through the image (as in 2D convolution). A pixel in the original image (either 1 or 0) will be considered 1 only if all the pixels under the kernel is 1, otherwise it is eroded (made to zero). So what happens is that, all the pixels near boundary will be discarded depending upon the size of kernel. So the thickness or size of the foreground object decreases or simply white region decreases in the image. It is useful for removing small white noise, detach two connected objects etc.
7. Dilate: It is just opposite of erosion. Here, a pixel element is '1' if at least one pixel under the kernel is '1'. So it increases the white region in the image or size of foreground object increases. Normally, in cases like noise removal, erosion is followed by dilation. Because, erosion removes white noises, but it also shrinks our object. So we dilate it. Since noise is gone, they won't come back, but our object area increases. It is also useful in joining broken parts of an object.
8. Find contours: Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same color or intensity. The contours are a useful tool for shape analysis and object detection and recognition. Here the system tracks the largest contour and relay centroid of contour through a socket connection to the microcontroller unit.

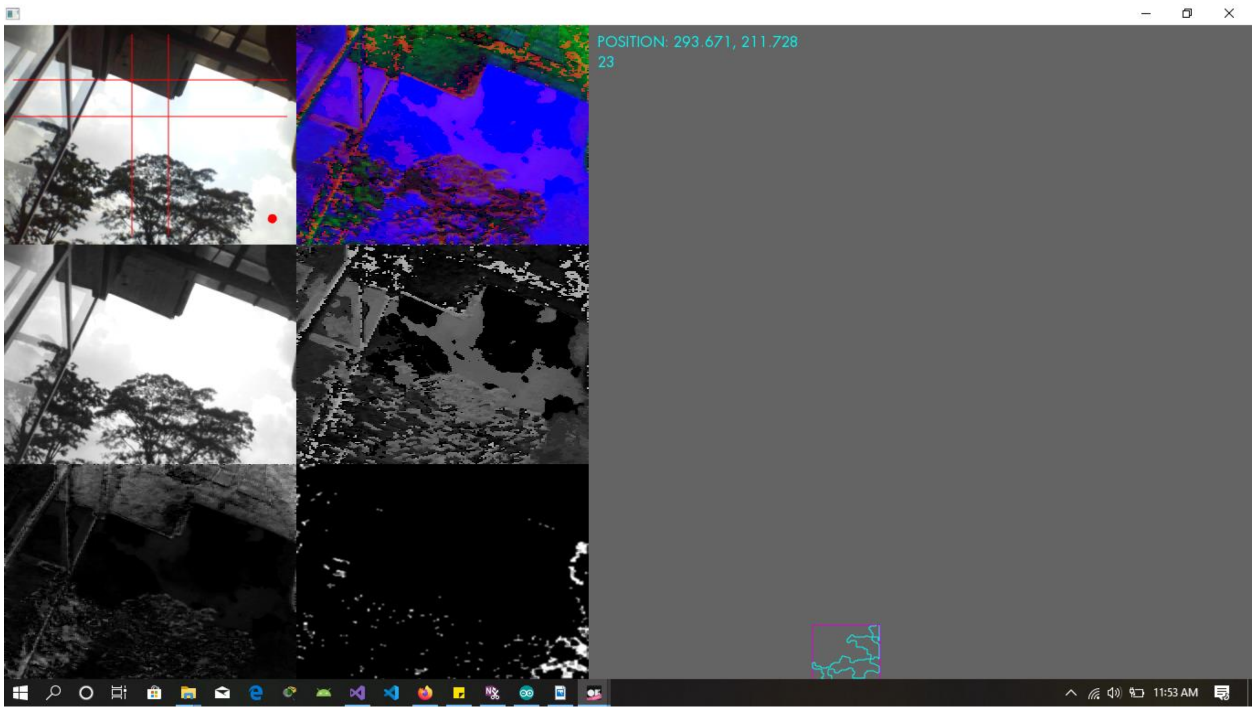

POSITION: 293.671, 211.728
23

Figure 15 Navigation Panel

3.1.4 Navigation System

The system provides the user a graphic user interface to control the robot; however, our primary goal is to let the robot respond to the gesture, since this is what is driving the robot it is crucial for us to figure out a suitable navigation algorithm to
guide the robotic car path . In this part of the report, we will explain the navigation system of our robot.

To give the robotic car normal direction paths cars use for navigation, a navigation grid with nine (9) segments is designed and used to define a region of space for the object position at every instance from gesture capture (head movement). The robot will use the position of the object in the screen to adjust the its moving direction.

Once the human forehead with a red color object attached to it falls in any of the nine coordinates its moves accordingly to the direction of the coordinate. The nine coordinates and directions are forward, reverse, left, right and stop directions**.**

3.1.5 Wireless Communication

This entails the gesture control using communicating with the robotic car via Wi-Fi. Specifically, socket is used for the wireless communication.

A **socket** is one end-point of a two-way communication link between programs running on the network. A socket connection is the most basic low-level reliable communication mechanism between a wireless device and a remote server or between two wireless devices.

To use a socket, the sender (gesture control system) and receiver (the controller board) that are communicating must first establish a connection between their sockets. One will be listening for a request for a connection, and the other will be asking for a connection. Once two sockets have been connected, they are used for transmitting data from the gesture system to the controller board direction.

3.1.6 User Interface

The user interface is the place where the user interacts with the robot. It provides the user a graphic user interface to control the robot or let the user know what happens when the robot is automatically undertaking the sample return mission. The GUI is programmed using C++ to guarantee the portability of the program. The interface consists of two windows, the first window combines the navigation panel and the data display while the other window displays the video field from the camera on the robotic car.

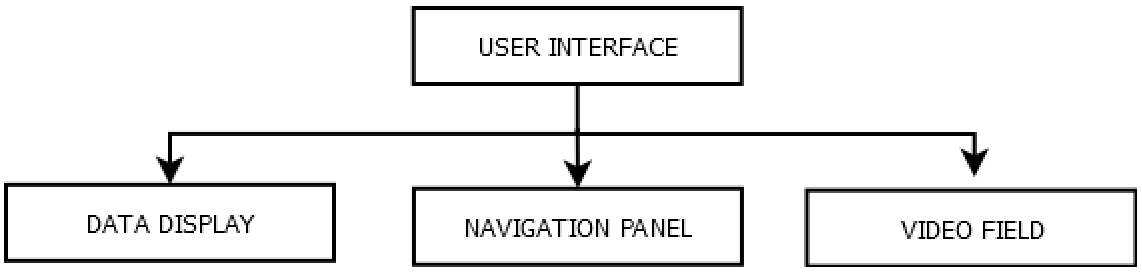


Figure 16 User Interface

The navigation panel displays the object tracking and positioning processes; it also shows the direction grids with different coordinates that classifies the object position.

The video field is displayed on a web browser; this windows displays the environment from the car's view. The major purpose of this field is for feedback during remote control, this helps to control environmental factors in the operating area of the system in case of obstacles in the car navigation path to aid driving.

3.2 CAR SYSTEM

3.2.1 Controller Board (ESP8266)

The embedded controller is programmed using Arduino IDE. This mechanism establishes a connection between the motor driver and object tracking for navigation of the robotic car using. It receives the position data from the tracking software.

3.2.2 Drive System and Motor Driver (ESP 12E Motor Shield)

The main IC on the ESP 12E motor shield board is the L293D (H-bridge). The drive system work as:

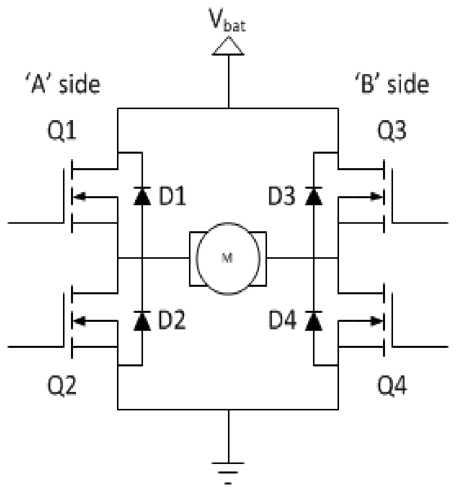


Figure 17 H-bridge b

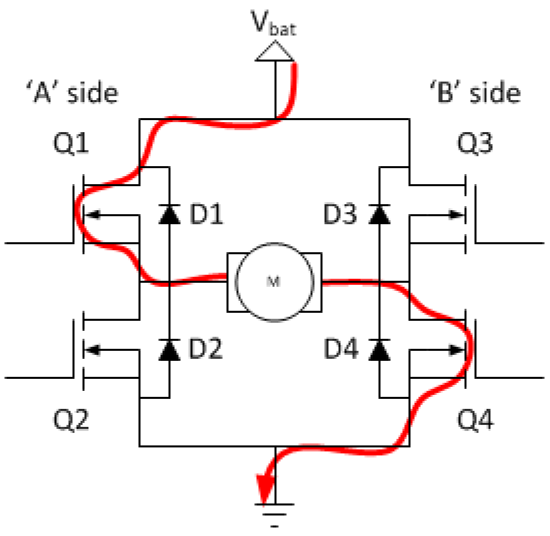


Figure 18 H-bridge c

The basic operating mode of an H-bridge is fairly simple: if Q1 and Q4 are turned on, the left lead of the motor will be connected to the power supply, while the right lead is connected to ground. Current starts flowing through the motor which energizes the motor in (let's say) the forward direction and the motor shaft starts spinning. If Q2 and Q3 are turned on, the reverse will happen, the motor gets energized in the reverse direction, and the shaft will start spinning backwards.

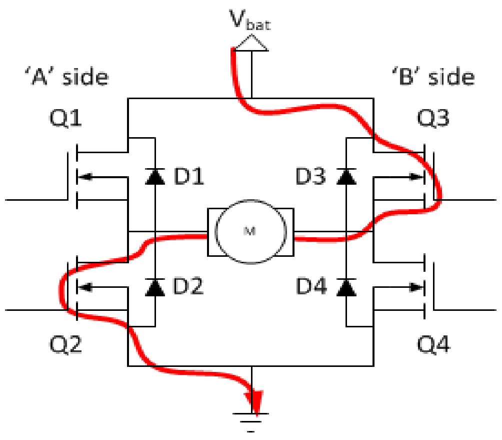


Figure 19 H-bridge d

3.2.3 Camera Integration

The OV2460 Arduino camera used for video field as feedback is integrated with the ESP-32 CAM module for its proper operation. The camera module is programmed with an Arduino Uno programmer board using Arduino IDE as the environment for coding and upload of program.

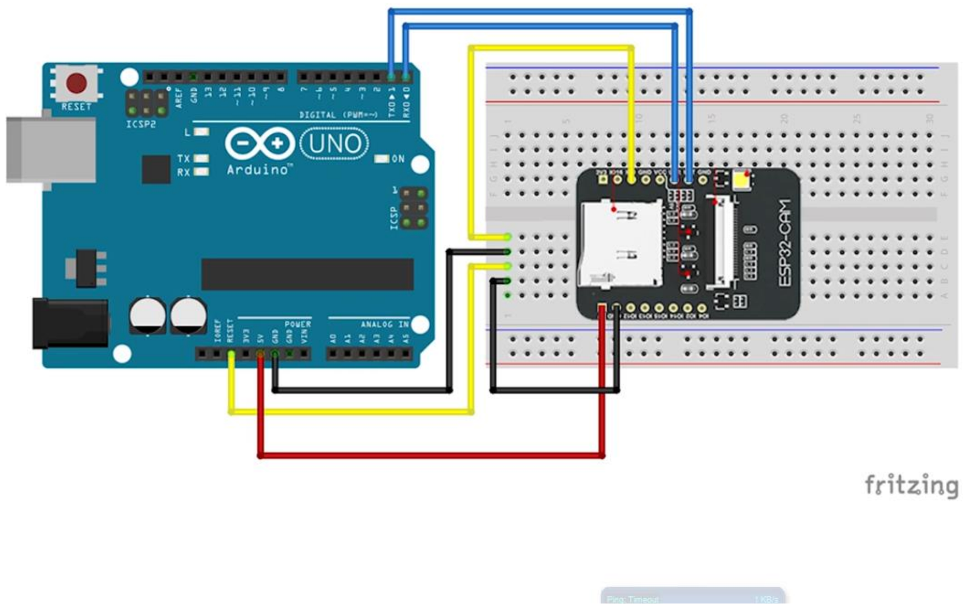
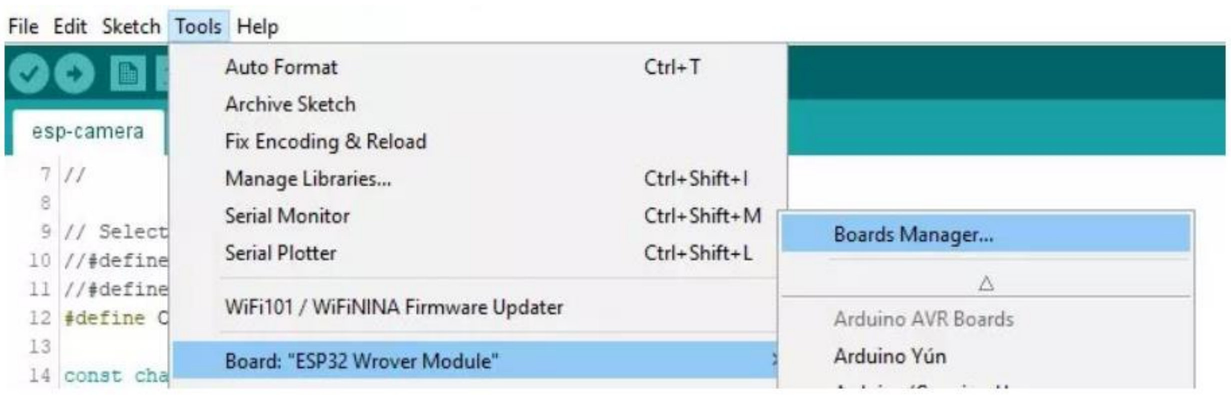


Figure 20 Programming with an Arduino UNO

While the esp32-cam module is connected to the Arduino Uno board, some necessary integration is to be made on the Arduino IDE. They are;

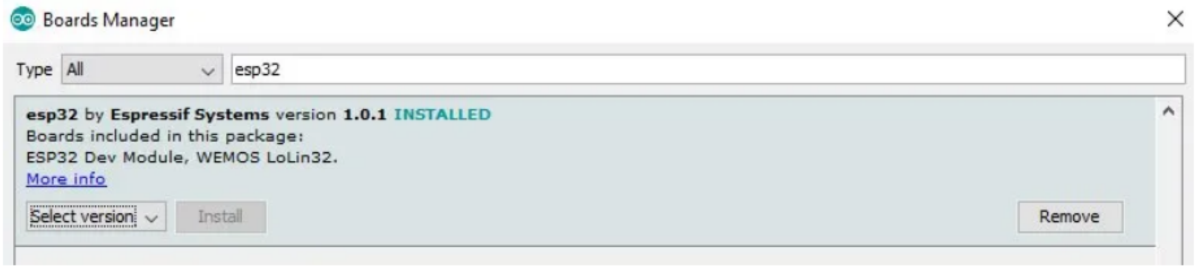
1. First select the board type in tool menu and search for esp8266


File Edit Sketch Tools Help
esp-camera
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater
Board: "ESP32 Wrover Module"
Boards Manager...
Arduino AVR Boards
Arduino Yún

```
//

// Select
//#define
//#define
#define C

const cha
```

Figure 21 Cam Module1

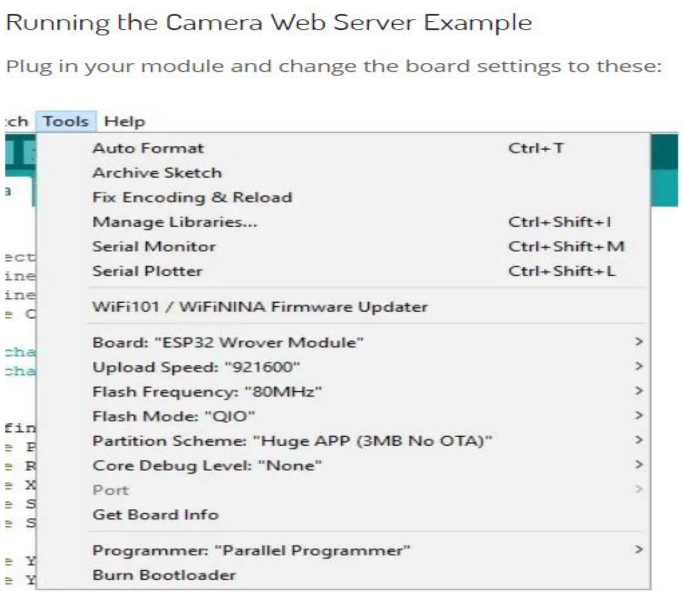
2. Then the ESP board is installed



esp32 by **Espressif Systems** version **1.0.1** INSTALLED
Boards included in this package:
ESP32 Dev Module, WEMOS LoLin32.
More info

Figure 22 Cam Module2

3. Then the camera web server is setup and ran


Running the Camera Web Server Example

Plug in your module and change the board settings to these:

Tools
Help
Auto Format Ctrl+T
Archive Sketch
Fix Encoding & Reload
Manage Libraries... Ctrl+Shift+I
Serial Monitor Ctrl+Shift+M
Serial Plotter Ctrl+Shift+L
WiFi101 / WiFiNINA Firmware Updater

Board: "ESP32 Wrover Module"
Upload Speed: "921600"
Flash Frequency: "80MHz"
Flash Mode: "QIO"
Partition Scheme: "Huge APP (3MB No OTA)"
Core Debug Level: "None"

Programmer: "Parallel Programmer"

Figure 23 Cam Module3

4. Open the Sketch by navigating File > Examples > ESP32 > Camera > Camera Webserver:

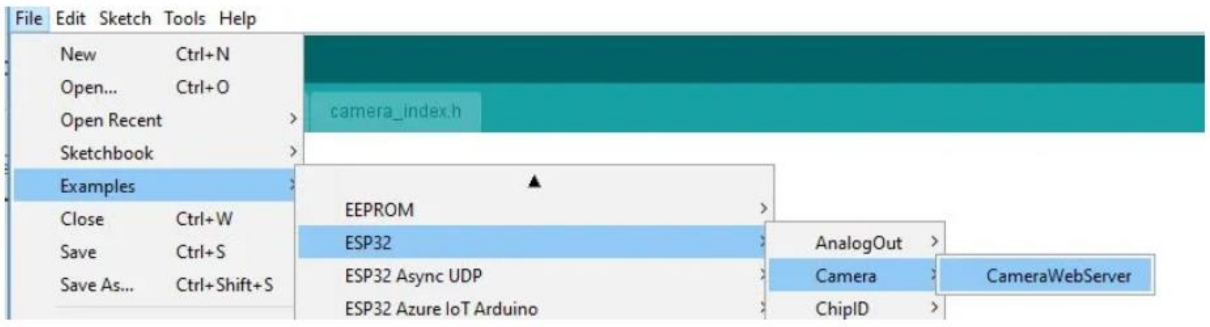


Figure 24 Cam Module 4

5. After the sketch is uploaded to the ESP Cam module, its sends the camera feed via a http socket, which is displayed on a web browser on a device connected to the camera module’s IP address.

CHAPTER FOUR

CONSTRUCTION, TESTING AND RESULTS

This section describes in details all the steps taken in the construction of this project, the necessary test carried out at different stages of the design and the final result after completion of construction and testing.

4.1 CONSTRUCTION

4.1.1 Robotic Car

The main structure of the robotic car is the car chassis, every components including motor driver, controller board and camera sits on the car chassis. The chassis is made of plastic which is good enough for the design since it's a prototype. The chassis was put together as follows;

1. The chassis frame was unpacked and unwrapped.

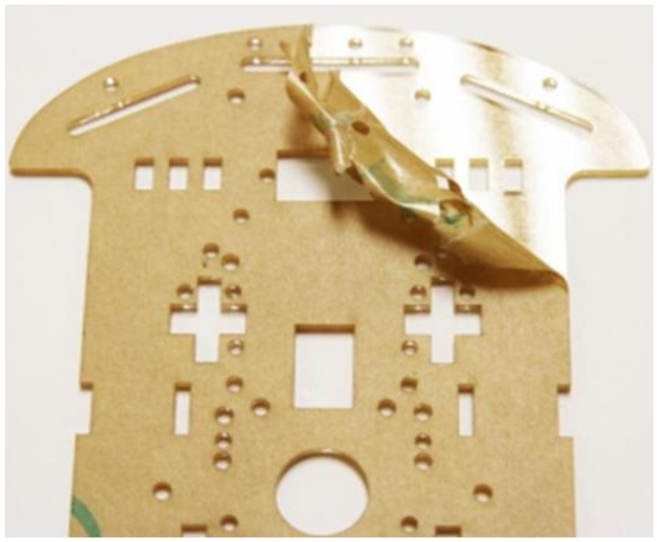


Figure 4.1.1a Car Chassis 1

2. From chassis instruction manual two frames were fixed to the base frame to hold the two DC motors and their wheels.

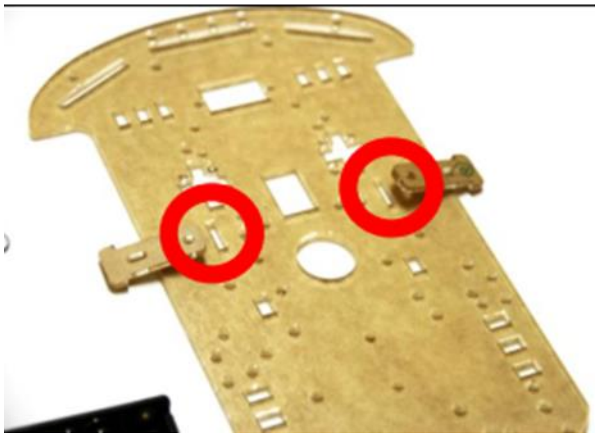
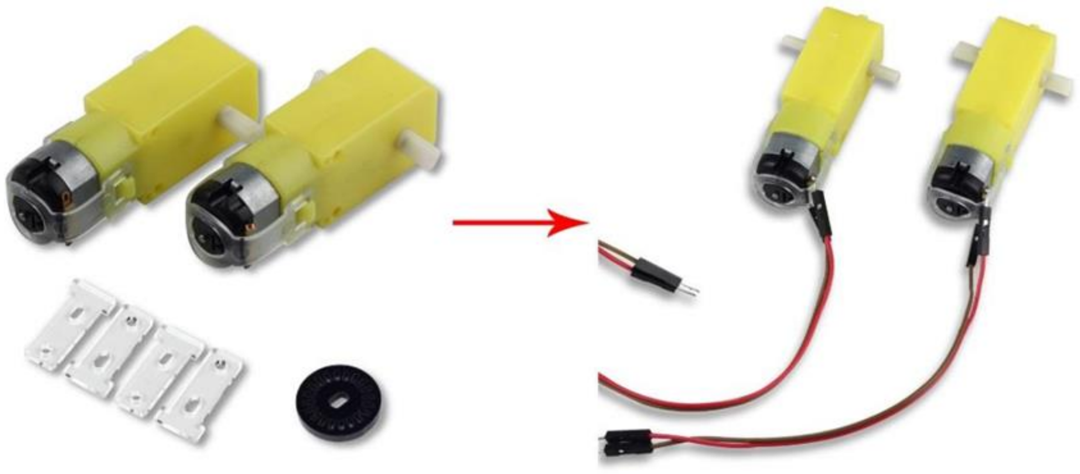
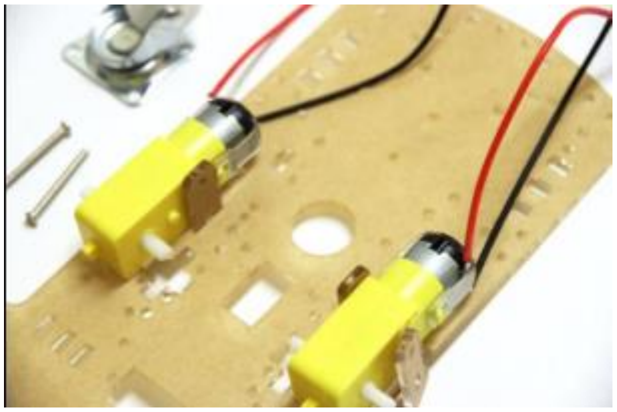


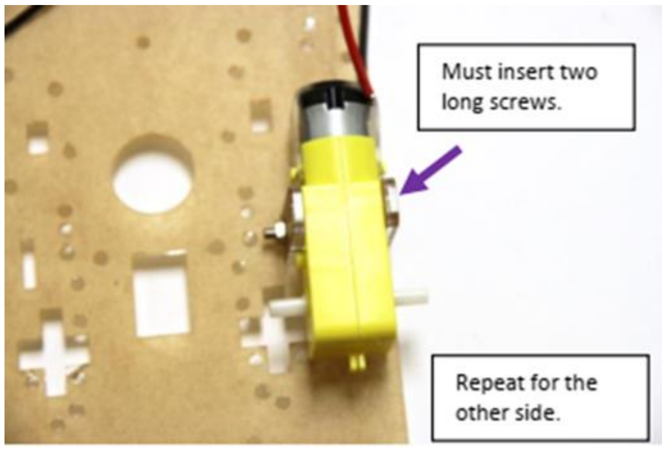
Figure 4.1.1b Car Chassis

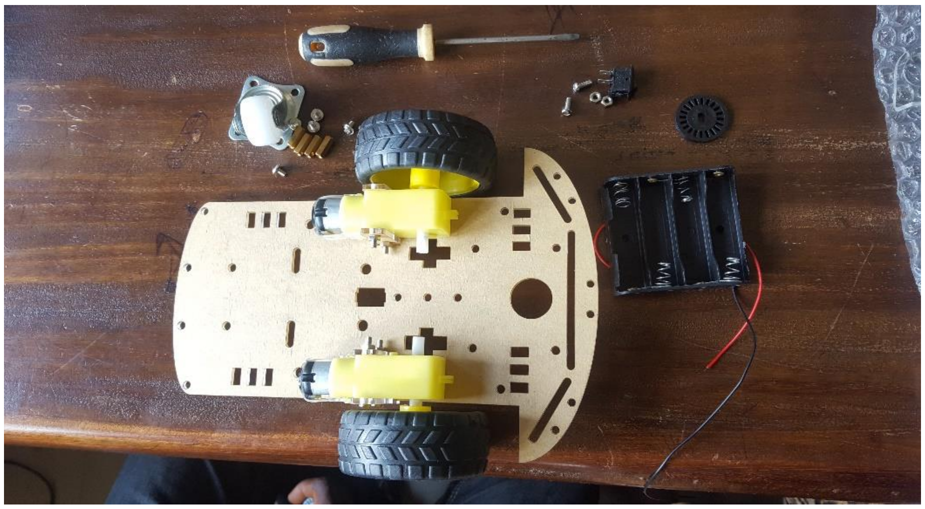
3. The two DC motors casings were fixed to them respectively using m3 sized screws provided from the package, also two wires were soldered to the motor to power them. After connection each motor was tested by powering it with a nine volts battery.



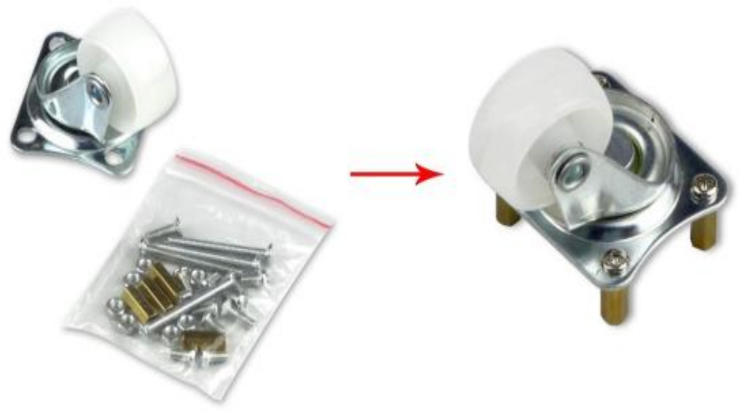
4. Now the tested motors and wheels are fixed to the chassis frame.

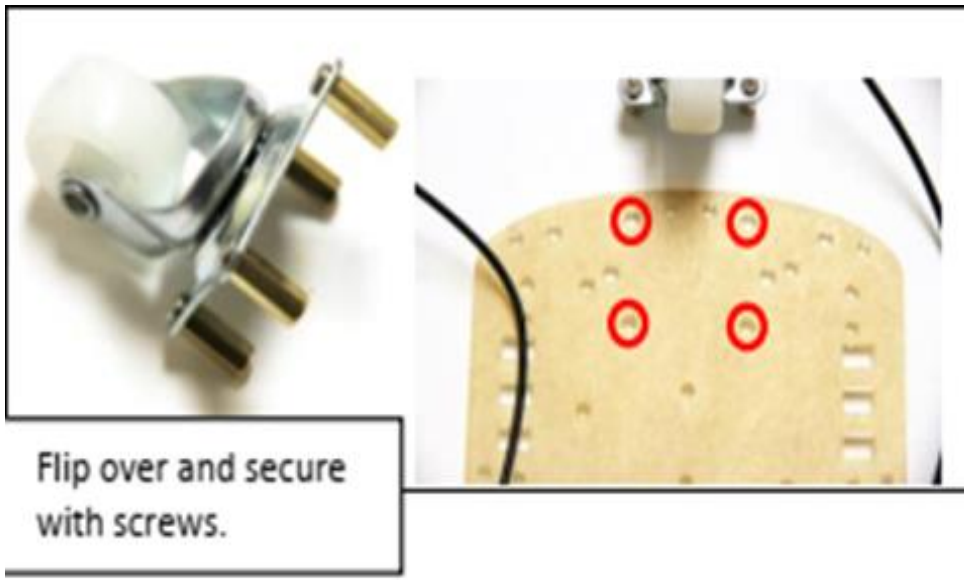



Must insert two long screws.
Repeat for the other side.

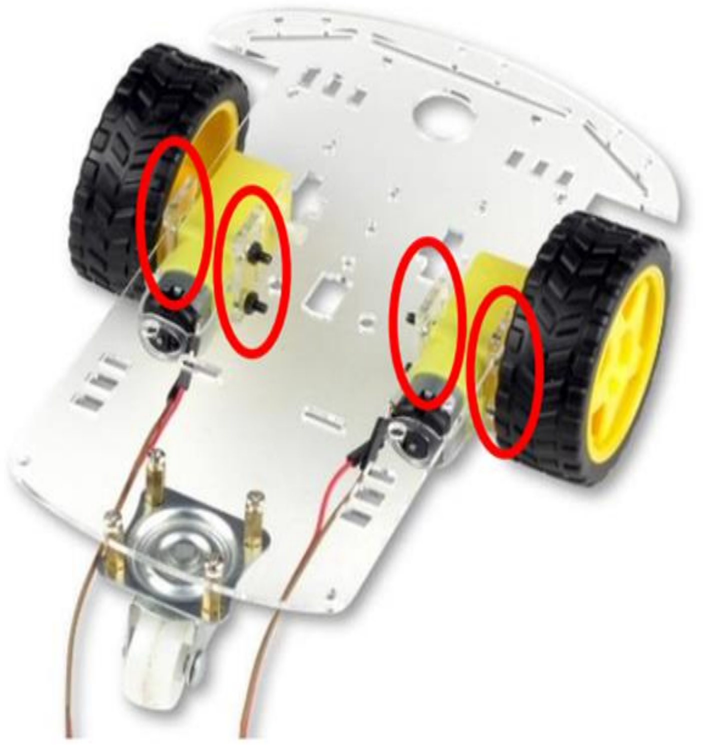


5. A steering wheel is fixed to the frame since the car design is a two-wheeled drive, a front steering wheel is needed for balance. The wheel was fixed and assembled as shown in the images below:

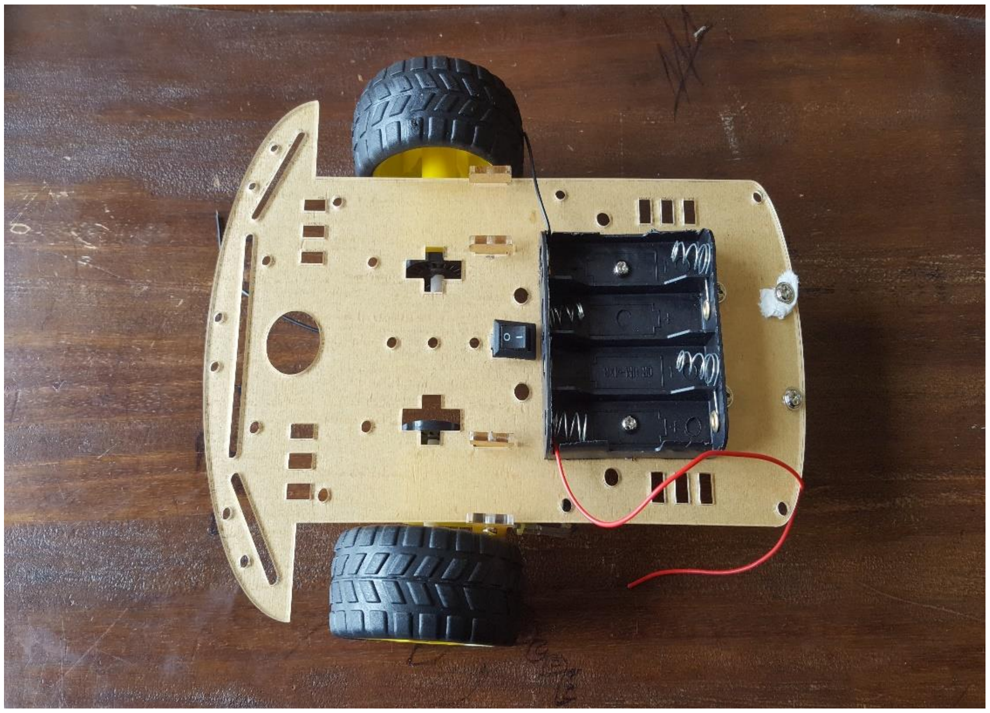




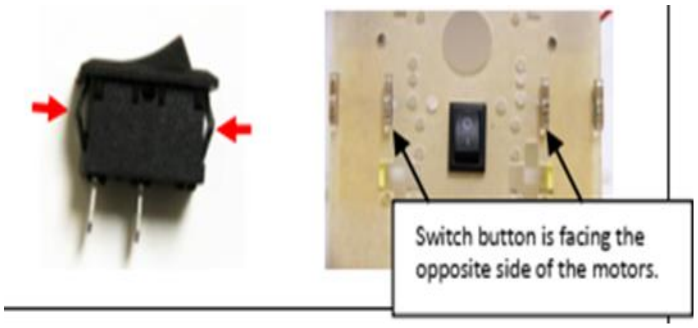
6. Lose screws were tightened;



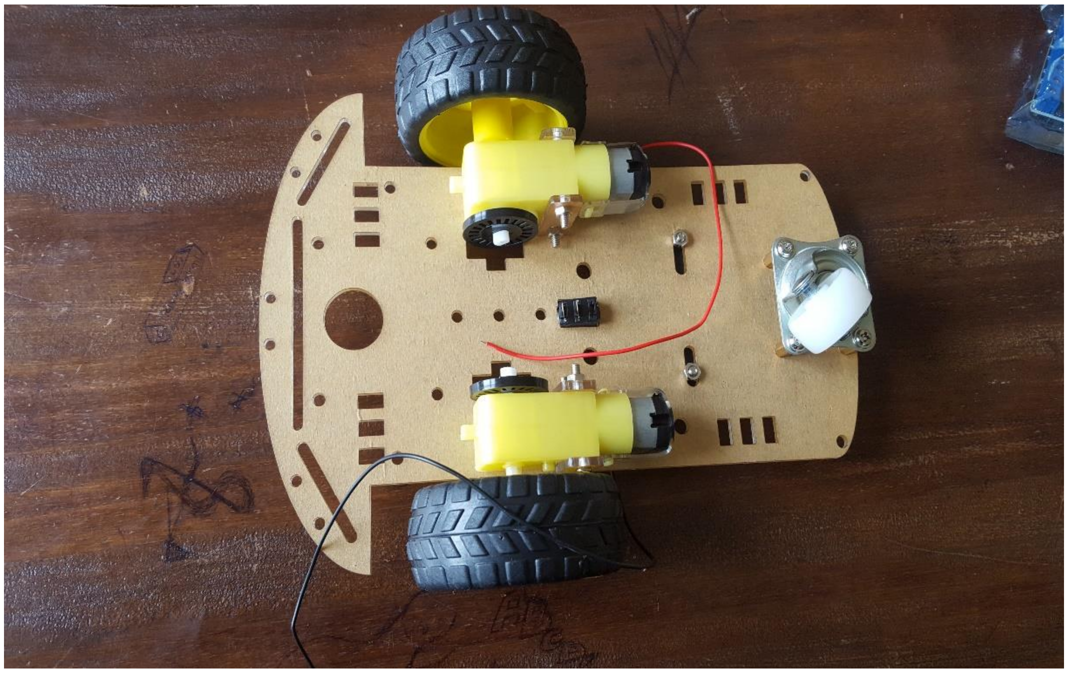
7. The battery pack is fixed on the top frame. The pack is capable of containing four AA sized batteries to power the whole car system. Two M3 bolts and nuts were used to hold it firm on the frame.



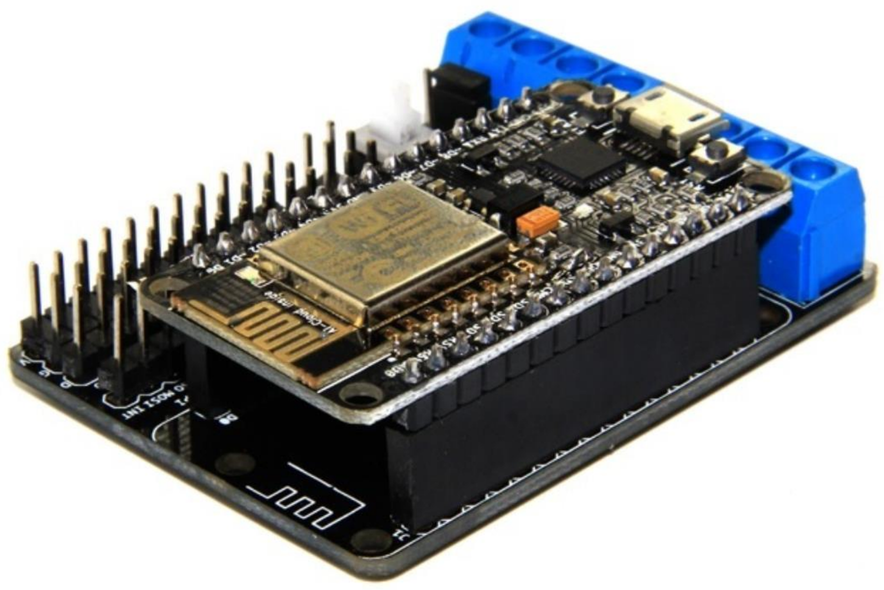
8. The switch button to turn on and off the power source from the battery pack is fixed on the top frame for easy control.



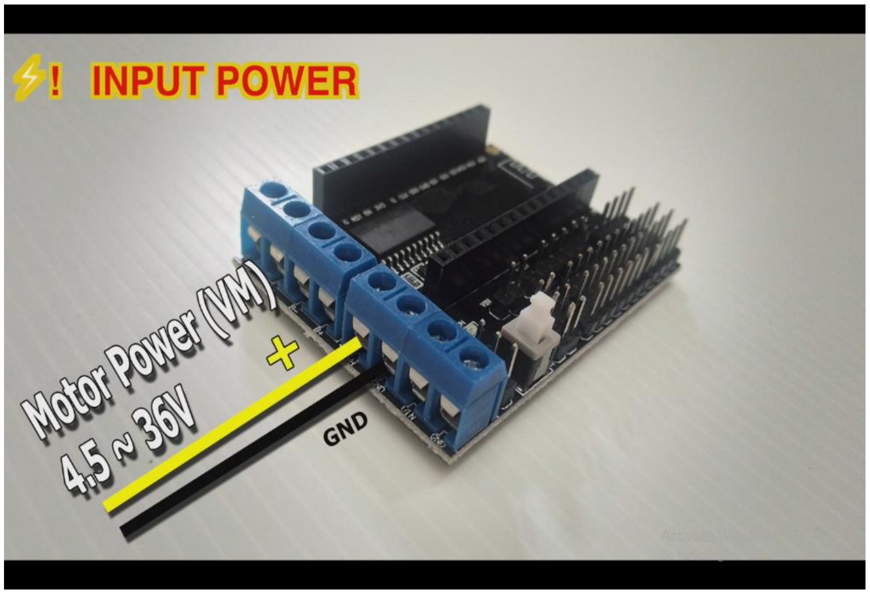
Now the car is fully assembled;



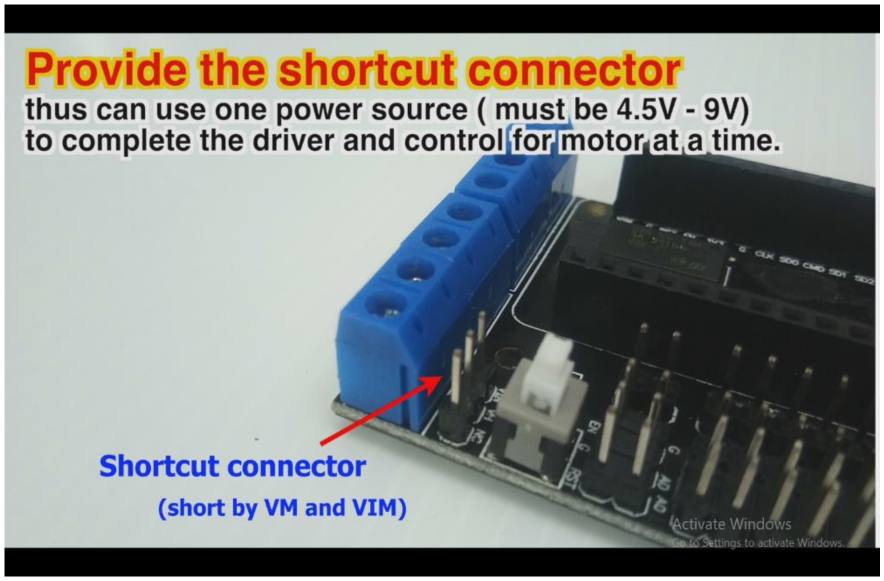
The ESP 8266 controller board is coupled with the ESP12E motor shield to form a unit as shown below;



Now the unit becomes the main controller board, and they have a common power source which makes the connections neater. Connecting wires were connected from output ports A+, A-, B+, B- to the two DC motors.



The controller board is connected to the power source using the ESP12E motor shield common power pin VM and VIM to power both controller and motors.


Provide the shortcut connector
thus can use one power source (must be 4.5V - 9V)
to complete the driver and control for motor at a time.
Shortcut connector
(short by VM and VIM)
Activate Windows
Go to Settings to activate Windows.

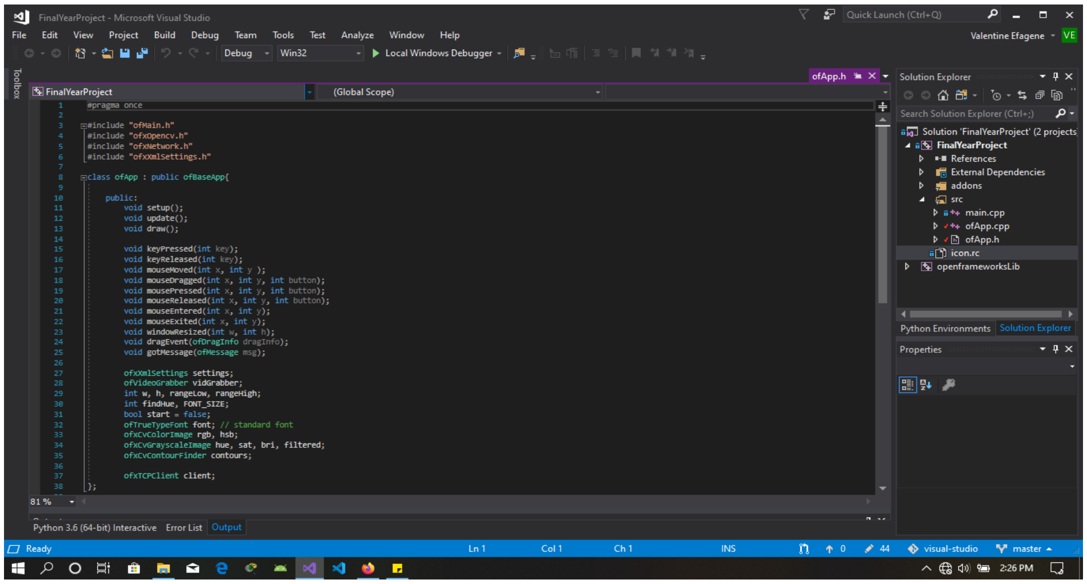
The Complete car system:

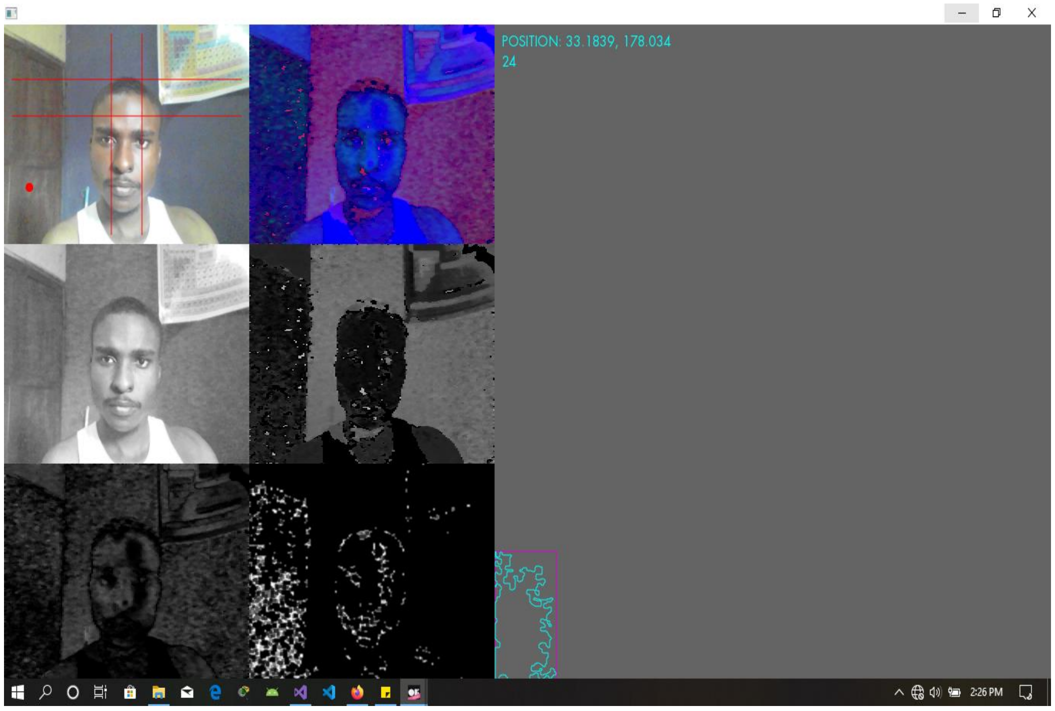
4.1.2 Gesture Control System

A detailed design and architecture of the gesture control system software is done in the methodology; this section only contains the details of how it was implemented using different softwares.

The whole gesture control system sits on the laptop with a webcam. Basically any laptop with standard specification can run the program. For this project a HP Elitebook 840G3 with 8gb RAM, 500gb hard disk storage, 2.5ghz processor speed was used to design and implement the gesture control program. The system program is compiled in an execution file with other necessary openframe works library files in a setup folder which can run on any windows operating system laptop.

The object tracking system was designed using openCV library with open openframeworks as the underlying design for the program. All this is designed and built using Visual Studio 20 IDE, and specifically with C++ programming language


```cpp
#pragma once

#include "ofMain.h"
#include "ofxOpencv.h"
#include "ofxNetwork.h"
#include "ofxXmlSettings.h"

class ofApp : public ofBaseApp{

    public:
        void setup();
        void update();
        void draw();

        void keyPressed(int key);
        void keyReleased(int key);
        void mouseMoved(int x, int y );
        void mouseDragged(int x, int y, int button);
        void mousePressed(int x, int y, int button);
        void mouseReleased(int x, int y, int button);
        void mouseEntered(int x, int y);
        void mouseExited(int x, int y);
        void windowResized(int w, int h);
        void dragEvent(ofDragInfo dragInfo);
        void gotMessage(ofMessage msg);

        ofxXmlSettings settings;
        ofVideoGrabber vidGrabber;
        int w, h, rangeLow, rangeHigh;
        int findHue, FONT_SIZE;
        bool start = false;
        ofTrueTypeFont font; // standard font
        ofxCvColorImage rgb, hsb;
        ofxCvGrayscaleImage hue, sat, bri, filtered;
        ofxCvContourFinder contours;

        ofxTCPClient client;
};
```


```
POSITION: 33.1839, 178.034
24
```

4.2 BILL OF ENGINEERING MEASUREMENT

This present a summary of the budget report of all the component parts, materials, equipment and research used for the construction of the project in a tabular form stating the parts name, unit cost, quantity used and total cost.

Table 4.1: Bill of Engineering Measurement

S/N	ITEMS	QUANTITY	UNIT PRICE (₦)	TOTAL PRICE (₦)
1.	Robotic Car Chassis	1	5,000	5,000
2.	DC Motors	2	500	1,000
3.	Servo Motor	1	500	500
4.	ESP12E Motor Shield	1	1000	1000
5.	ESP32 CAM BOARD	1	2000	2000
6.	OV2460	1	2000	2000
7.	ESP8266 Controller	1	2,000	2,000
8.	Subscription	-	5000	5000
9.	Printing	-	5,000	5,000
10.	Miscellaneous	-	2,000	2,000
	TOTAL			25,500

4.3 TESTING

In the construction and implementation of this project each unit was tested to ensure that they met the required specification stated by the project design and objectives.

4.3.1 Robotic Car System

The robotic car system is the hardware unit of this project. Each parts and segment of the car system was tested individually using the proper tools.

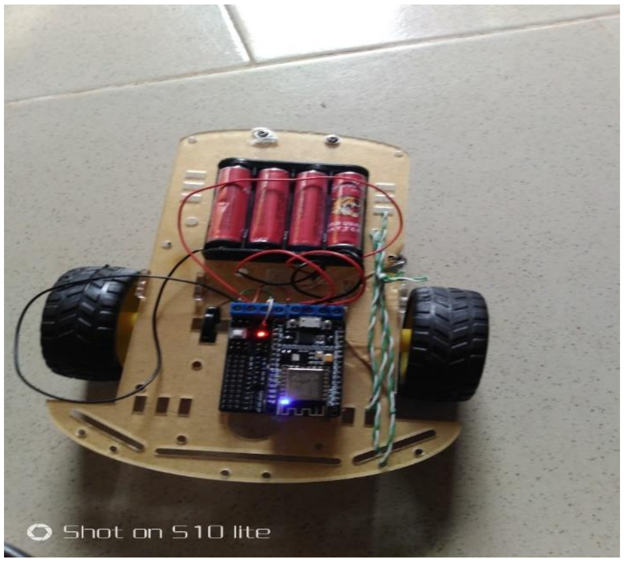
The Controller Board and ESP12E motor shield- The ESP 8266 controller was tested to ensure its working properly by uploading sample codes written with the Arduino IDE. The result showed the board was working properly. The output and input pins of the controller board and ESP 12E were tested using a digital multi-meter to check for of their voltages when connected to power met their individual datasheet specifications.

The Drive System- The drive system consists of two wheels connected to two DC motors, a front steering wheel. This unit was tested as a whole and in parts. The individual DC motors were tested with 9volts battery to check for movement. As a whole the drive system was tested with a joystick to ensure the movement of the car and its response to direction change.

The OV2460 Camera and ESP 32 Cam Module – The two components are designed to work together as a unit. The unit was tested with a sample code to check the camera quality and its video field functionality. There was a problem encountered while integrating the camera and its module. After debugging, the error was traced to a library problem and it was rectified.

The power supply for the project is solely a DC source using four AA sized batteries connected in series on a battery pack placed on the car chassis. With a digital multi-meter, the output voltage read 6.1 volts.

The Car system is finally tested after it was assembled together and its moved around an open space in all directions.



4.4 RESULT

The criteria for judging this project is to see if the robotic car would respond to the gesture (head movement) being made. The result for the system tests are shown in the table below;

S/N	TEST	PROCEDURE	OBSERVATION
1.	The Drive System	The drive system is powered on and its receives navigation code from the object tracking system	Its observed that the car moved according to the navigation codes received
2.	The Camera and ESP32 Cam Module	The camera is powered on.	The video field is sent on a http socket which can be retrieved from a browser window
3.	The Object Tracking System	A tag is attached to the forehead and	The gesture is captured with the object tracking system and object position moves along with the head movement

4.5 TIMELINE

This section shows a detailed draft of the whole project timeline from the research stage to the final stage.

Table 4.5: Project Timeline

S/N	**EXPECTATION**	**ACTIVITY**	**WEEKS**
1.	Detailed Research On Project	Online Research	8 Weeks (October - November)
2.	Scope of Study	Feasibility Study and Defining Scope of Study	8 Weeks (December-January 2019)
3.	Project Methodology Definition	Testing different method Of object tracking with openCV	6 Weeks (April – May 14th)
4.	Hardware Integration Mechanism	Selection of chassis, controller and testing	2 Weeks (May 15-27)
5.	Project Proposal Draft	Learning & Understanding of Write-up format	1Week(May 29-June 6)
6.	Project Proposal	Proposal Write-up and Submission for approval	2 Weeks(June 7-13)
7.	Software Integration	Gesture-Control System Development	4 Weeks (August 1-30)
8.	Chassis	Assembling of Chassis	1 Week (Sept 4-11)
9.	Extensive Testing and Write-up		5 Weeks (Oct 3- Oct 31, Dec 6-Dec 13)

CHAPTER FIVE

CONCLUSION, LIMITATION AND RECOMMENDATION

5.1 CONCLUSION

The gesture controlled robotic car was design, constructed and implemented. This project uniqueness is the response of the robotic car to the gesture being made, and this was possible as a result of the accuracy of the color filtering algorithm (chosen for object tracking which was used to capture the human gesture) in an environment with little or no color noise.

In conclusion, this project was successful as the main objective of designing and constructing the gesture controlled robotic car was achieved after various troubleshooting and testing. This project is cost-effective, reliable and easy to operate with proper guidance.

5.2 LIMITATION

In the course of this project, there were some restrictions and drawbacks in the implementation. Some of these problems encountered are;

i. The vision algorithm (color filtering) implemented in this project performance was affected greatly in an environment with much color noise. In this environment the accuracy of the detection of gesture capture was reduced.

ii. The material used for the car chassis is plastic and this made it susceptible to breakage in an outdoor environment. During testing, we were very careful to clear the environment of any obstacle with excessive weight to avoid damage to the chassis in case of collision. Without proper supervision the car chassis probability of getting damaged is high.

iii. The size of the whole system is a limitation, a laptop is used to implement the gesture control system which affects portability and mobility of the project.

iv. For the navigation system it is quite difficult to keep the human head movement in a steady position over a period of time. This instability affects the steering of the robotic car while in motion.

5.3 RECOMMENDATION

From the processes and work carried out in the implementation of this project, some recommendations are proffered despite having met all of our aims and objectives because there is much room for future improvement. The recommendations are divided into two categories they are;

- Mechanical Recommendation
- Software Recommendation

5.3.1 Mechanical Recommendation

i. A four wheeled car chassis with a rugged and robust frame should be used for resilience and stability in an outdoor environment.

ii. A gripper should be added to the car structure to allow the robot to be able to grab objects for autonomy.

5.3.2 Software Recommendation

i. Two or more algorithm should be implemented together to improve user experience and provide a more accurate detection of gesture.

   The object tracking system could switch between the two vison algorithm to detect target object accurately in a situation when color noise is too large, the color tracking algorithm is ignored and another algorithm is then used for detection.

References

Dokic, A. P. (2013). Wheelchair Control by Head Motion, Serbian Journal of Electrical Engineering, Vol 1, No1, 135-151.

Finnegan, D. J. (2012). Object Detection and Tracking in Images and Point Clouds.

Gartner. (2019). Retrieved from www.gartner.com

Instructables. (2017). Retrieved from www.instructables.com

JA, A. (2002). Critical Considerations for Human-Robot Interface Development", Proceedings of 2002 AAAI Fall Symposium, Rochester Institute of Technology Rochester, New York, www.aaai.org .

K., A. M. (2017). OpenCV-Python Tutorials Documentation, Release 1 .

Kreinin, Y. (13 October 2009). "Defective C++".

Marr, D. (1982). Vision: A Computational Investigation into the Human Representation and Processing of Visual Information'', Freeman, San Francisco).

Santos, R. (2019, 8 9). *$7 ESP32-CAM with OV2640 Camera*. Retrieved from Make Advisor: https://makeradvisor.com/esp32-cam-ov2640-camera/

Saurabh Kandalkar, P. K. (May 2015). Arduino Based Head Gesture Controlled Robot Using Wireless. *Int. Journal of Engineering Research and Applications*, 1-3.

Stroustrup, B. (17 February, 2014). "C++ Applications". stroustrup.com. Retrieved 5 May 2014.

Stroustrup, B. (2014). *"Lecture:The essence of C++. University of Edinburgh". Retrieved 12 June 2015.*

Uctronics. (2017). *Arduino Smart Robot Car Kit.* www.uctronics.com.

What is Color Space. (2016). Retrieved from ArcSoft: http://www.arcsoft.com/topics/photostudio-darkroom/what-is-color-space.html

Wikipedia. (2019, December 11). *Article*. Retrieved from Wikipedia.com: https://en.wikipedia.org/wiki/Microsoft_Visual_Studio

APPENDIX I

Meta-Analysis Table

Table 3- Meta-Analysis Table

S/N	AUTHOR’S NAME & YEAR	TITLE OF WORK	METHODOLOGY	LIMITATION
1.	Tianci Zhao and Peng Ren	Intelligent Vision-Driven Robot for Sample Detection & Return	Video analysis with color filtering and Haar cascading classification methods is used for incoming feeds from the camera. A robust mobile robotic chassis with a robotic arm is used.	Wired connection between the system and robot limits detection range
2.	Daniel J. Finnegan, 2012	Object Detection and Tracking in Images and Point Clouds		The tracking system cannot indicate the pose of any object it detects and its liable to loss of detection with any foreground object
3.	Vladimir Kravtchenko, 1992	Tracking Color Objects In Real-Time.	Color Clustering in color space	Camera movement

				caused change in object dynamic.
4.				
5.	Saurabh Kandalkar, Pratik Kadam	Arduino Based Head Gesture Controlled Robot Using Wireless Communication		
6.				

APPENDIX II

Object Tracking Code

Main

```
#include "ofMain.h"
#include "ofApp.h"

//========================================================================
====
int main( ){
	ofSetupOpenGL(1024,768,OF_WINDOW);			// <-------- setup the GL
context

	// this kicks off the running of my app
	// can be OF_WINDOW or OF_FULLSCREEN
	// pass in width and height too:
	ofRunApp(new ofApp());

}
```

Header

```
#pragma once

#include "ofMain.h"
#include "ofxOpencv.h"
#include "ofxNetwork.h"
#include "ofxXmlSettings.h"

class ofApp : public ofBaseApp{

	public:
		void setup();
		void update();
		void draw();

		void keyPressed(int key);
		void keyReleased(int key);
		void mouseMoved(int x, int y );
		void mouseDragged(int x, int y, int button);
		void mousePressed(int x, int y, int button);
		void mouseReleased(int x, int y, int button);
		void mouseEntered(int x, int y);
		void mouseExited(int x, int y);
		void windowResized(int w, int h);
		void dragEvent(ofDragInfo dragInfo);
		void gotMessage(ofMessage msg);
```

```
                ofxXmlSettings settings;
                ofVideoGrabber vidGrabber;
                int w, h, rangeLow, rangeHigh;
                int findHue, FONT_SIZE;
                bool start = false;
                ofTrueTypeFont font; // standard font
                ofxCvColorImage rgb, hsb;
                ofxCvGrayscaleImage hue, sat, bri, filtered;
                ofxCvContourFinder contours;

                ofxTCPClient client;
};
```

Main Color Tracking Code

```
#include "ofApp.h"

//--------------------------------------------------------------
void ofApp::setup(){
	w = 320;  // try to grab at this size.
	h = 240;
	FONT_SIZE = 12;
	findHue = 30;
	font.load("futura_book.otf", FONT_SIZE);

	if (!settings.loadFile("settings.xml")) {
		ofLogError() << "Couldn't load file";
	}

	string ip = settings.getValue("SETTINGS:IP", "127.0.0.1");
	int port = settings.getValue("SETTINGS:PORT", 12324);

	ofxTCPSettings setup(ip, port);
	client.setup(setup);
	client.setMessageDelimiter("\n");

	//get back a list of devices.
	vector<ofVideoDevice> devices = vidGrabber.listDevices();

	for (size_t i = 0; i < devices.size(); i++) {
		if (devices[i].bAvailable) {
			//log the device
			ofLogNotice() << devices[i].id << ": " <<
devices[i].deviceName;
		}
		else {
			//log the device and note it as unavailable
			ofLogNotice() << devices[i].id << ": " <<
devices[i].deviceName << " - unavailable ";
```

```
		}
	}

	vidGrabber.setDeviceID(0);
	vidGrabber.setDesiredFrameRate(60);
	vidGrabber.initGrabber(w, h, true);

	rgb.allocate(w, h);
	hsb.allocate(w, h);
	hue.allocate(w, h);
	sat.allocate(w, h);
	bri.allocate(w, h);
	filtered.allocate(w, h);
}

//--------------------------------------------------------------
void ofApp::update(){
	ofBackground(100, 100, 100);
	vidGrabber.update();

	if (vidGrabber.isFrameNew()) {

		rgb.setFromPixels(vidGrabber.getPixels());
		rgb.mirror(false, true);
		hsb = rgb;
		hsb.convertRgbToHsv();
		hsb.convertToGrayscalePlanarImages(hue, sat, bri);

		for (int i = 0; i < w * h; i++) {
			filtered.getPixels()[i] =
ofInRange(hue.getPixels()[i], findHue, findHue) ? 255:0;
		}

		filtered.flagImageChanged();
		filtered.blurGaussian(3);
		filtered.dilate();
		filtered.erode();
		contours.findContours(filtered, 50, w * h / 2, 1, false);
	}
}

//--------------------------------------------------------------
void ofApp::draw(){
	const int W = 320, H = 240, MARGIN = 10, SQUARE_WIDTH = 40;
	string action = "";
	rgb.draw(0, 0);
	hsb.draw(W, 0);
	bri.draw(0, H);
	hue.draw(W, H);
	sat.draw(0, 2 * H);
	filtered.draw(W, 2 * H);
	contours.draw(2 * W, 2 * H);
```

```
      ofSetColor(255, 0, 0);
      ofFill();
      float maxArea = 0;
      int index = 0;

      if(contours.nBlobs > 0) {
           for (int i = 0; i < contours.nBlobs; i++) {
                if (contours.blobs[i].area > maxArea) {
                     index = i;
                     maxArea = contours.blobs[i].area;
                }
           }

           // reverse
           if (contours.blobs[index].centroid.y < H / 4 &&
action.length() < 3) {
                action += "1";
           }

           //  forward
           if (contours.blobs[index].centroid.y > H * 0.5 - 20 &&
action.length() < 3) {
                action += "2";
           }

           // right
           if (contours.blobs[index].centroid.x > W / 2 + 0.5 *
SQUARE_WIDTH && action.length() < 3) {
                action += "3";
           }

           // left
           if (contours.blobs[index].centroid.x < W / 2 - 0.5 *
SQUARE_WIDTH && action.length() < 3) {
                action += "4";
           }

           // idle
           if (action == "") {
                action = "0";
           }

           ofSetColor(0, 255, 255);
           font.drawString("POSITION: " +
ofToString(contours.blobs[index].centroid.x) + ", "
                + ofToString(contours.blobs[index].centroid.y), 2 * W
+ MARGIN, 2 * FONT_SIZE);
           font.drawString(action, 2 * W + MARGIN, 3 * FONT_SIZE +
10);
           ofSetColor(255, 0, 0);
```

```
            ofCircle(contours.blobs[index].centroid.x, 
contours.blobs[index].centroid.y, 5);
            ofLine(MARGIN, H / 4, W - MARGIN, H / 4);
            ofLine(MARGIN, H * 0.5 - 20, W - MARGIN, H * 0.5 - 20);
            ofLine(W / 2 - 0.5 * SQUARE_WIDTH, MARGIN, W / 2 - 0.5 * 
SQUARE_WIDTH, H - MARGIN);
            ofLine(W / 2 + 0.5 * SQUARE_WIDTH, MARGIN, W / 2 + 0.5 * 
SQUARE_WIDTH, H - MARGIN);
            /*
            if (client.isConnected()) {

      client.sendRaw(std::to_string(contours.blobs[index].centroid.x) + 
" " +
                        std::to_string(contours.blobs[index].centroid.y) 
+ "\n");
            }*/

            if (client.isConnected() && start) {
                  client.sendRaw(action + "\n");
            }

            action = "";
      }

      ofSetColor(255, 255, 255);
}

//--------------------------------------------------------------
void ofApp::keyPressed(int key){

}

//--------------------------------------------------------------
void ofApp::keyReleased(int key){

}

//--------------------------------------------------------------
void ofApp::mouseMoved(int x, int y ){

}

//--------------------------------------------------------------
void ofApp::mouseDragged(int x, int y, int button){

}

//--------------------------------------------------------------
void ofApp::mousePressed(int x, int y, int button){
      int mX = x % w;
      int mY = y % h;
```

```
     findHue = hue.getPixels()[mY * w + mX];
     printf("%d", findHue);
     start = true;
}

//--------------------------------------------------------------
void ofApp::mouseReleased(int x, int y, int button){

}

//--------------------------------------------------------------
void ofApp::mouseEntered(int x, int y){

}

//--------------------------------------------------------------
void ofApp::mouseExited(int x, int y){

}

//--------------------------------------------------------------
void ofApp::windowResized(int w, int h){

}

//--------------------------------------------------------------
void ofApp::gotMessage(ofMessage msg){

}

//--------------------------------------------------------------
void ofApp::dragEvent(ofDragInfo dragInfo){

}
```

APPENDIX III

Controller Board Code

```
#include "ESP8266WiFi.h"


const char* ssid = "vee";

const char* password = "valentyne";

const uint16_t port = 12324;

WiFiServer server(port);

int direction = 0;

String s_dir = "";

int PWMA=5;//Right side

int PWMB=4;//Left side

int DA=0;//Right reverse

int DB=2;//Left reverse


void setup() {

  pinMode(PWMA, OUTPUT);

  pinMode(PWMB, OUTPUT);

  pinMode(DA, OUTPUT);

  pinMode(DB, OUTPUT);

  Serial.begin( 9600 );

  WiFi.begin( ssid, password );

  server.begin();
  while ( WiFi.status() != WL_CONNECTED ) {

    delay( 500 );

    Serial.println( "..." );

  }



  Serial.print( "WiFi connected with IP: " );

  Serial.println( WiFi.localIP() );

}



void loop() {

  WiFiClient client = server.available();



  if (client) {

    while (client.connected()) {

      while (client.available()>0) {

        s_dir = readLine(client);

        Serial.println(s_dir);

        direction = s_dir.toInt();



        switch (direction){

          case 0:

            s();
            break;
          case 1:
            b();
            break;
          case 2:
            f();
            break;
          case 3:
            tr();
            break;
          case 4:
            tl();
            break;
          case 13:
            br();
            break;
          case 14:
            bl();
            break;
          case 23:
            fr();
            break;
          case 24:
```

```
            fl();
            break;
           default:
            break;
        }
      }
    }

    client.stop();
    //Serial.println("Client disconnected");
  }
}

String readLine(WiFiClient client) {
  String data;
  boolean lineEnd = false;

  while (!lineEnd) {
    if (client.available()) {
      char c = client.read();

      if (c != '\n' && c != '\r') {
        data.concat(c);
      }

      if (c == '\n') {
        lineEnd = true;
      }
    }
  }

  return data;
}

// FORWARD AND RIGHT
void fr(){
  digitalWrite(PWMA, HIGH);
  digitalWrite(DA, LOW);

  digitalWrite(PWMB, LOW);
  digitalWrite(DB, LOW);
}

// BACK AND RIGHT
void br(){
  digitalWrite(PWMA, HIGH);
```

```
}

  digitalWrite(DA, HIGH);


  digitalWrite(PWMB, LOW);
  digitalWrite(DB, LOW);
}


// FORWARD AND LEFT
void fl(){
  digitalWrite(PWMA, LOW);
  digitalWrite(DA, LOW);


  digitalWrite(PWMB, HIGH);
  digitalWrite(DB, LOW);
}


// BACK
void b(){
  digitalWrite(PWMA, HIGH);
  digitalWrite(DA, HIGH);


  digitalWrite(PWMB, HIGH);
  digitalWrite(DB, HIGH);
}
```

```
// FORWARD
void f(){
  digitalWrite(PWMA, HIGH);
  digitalWrite(DA, LOW);

  digitalWrite(PWMB, HIGH);
  digitalWrite(DB, LOW);
}

// STOP
void s(){
  digitalWrite(PWMA, LOW);
  digitalWrite(DA, LOW);

  digitalWrite(PWMB, LOW);
  digitalWrite(DB, LOW);
}

// BACK AND LEFT
void bl(){
  digitalWrite(PWMA, LOW);
  digitalWrite(DA, LOW);
  digitalWrite(PWMB, HIGH);
  digitalWrite(DB, HIGH);
}

// TURN LEFT
void tl(){
  digitalWrite(PWMA, 450);
  digitalWrite(DA, HIGH);

  digitalWrite(PWMB, 450);
  digitalWrite(DB, LOW);
}

// TURN RIGHT
void tr(){
  digitalWrite(PWMA, 450);
  digitalWrite(DA, LOW);

  digitalWrite(PWMB, 450);
  digitalWrite(DB, HIGH); }
```

APPENDIX IV

Camera Integration Code