



People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research
University of Echahid Hamma Lakhdar - El Oued



Faculty of Technology
Department of Electrical Engineering
ACADEMIC MASTERS

Domain: Science and Technology
Department: Telecommunication
Specialty: Telecommunication Systems

Presented by:

✚ Bennour Brahim

✚ Melik Brahim

Entitled

**3D Camera Stabilization and Tracking System
with ESP32**

Dissertation submitted in partial fulfillment of the requirements for the Master's degree

Board of Examiners

Publicly Supported in: 23/09/2025

❖ **Chairman**

Pr. Bessous Nouredine

❖ **Examiner**

Dr. Labiod Chouaib

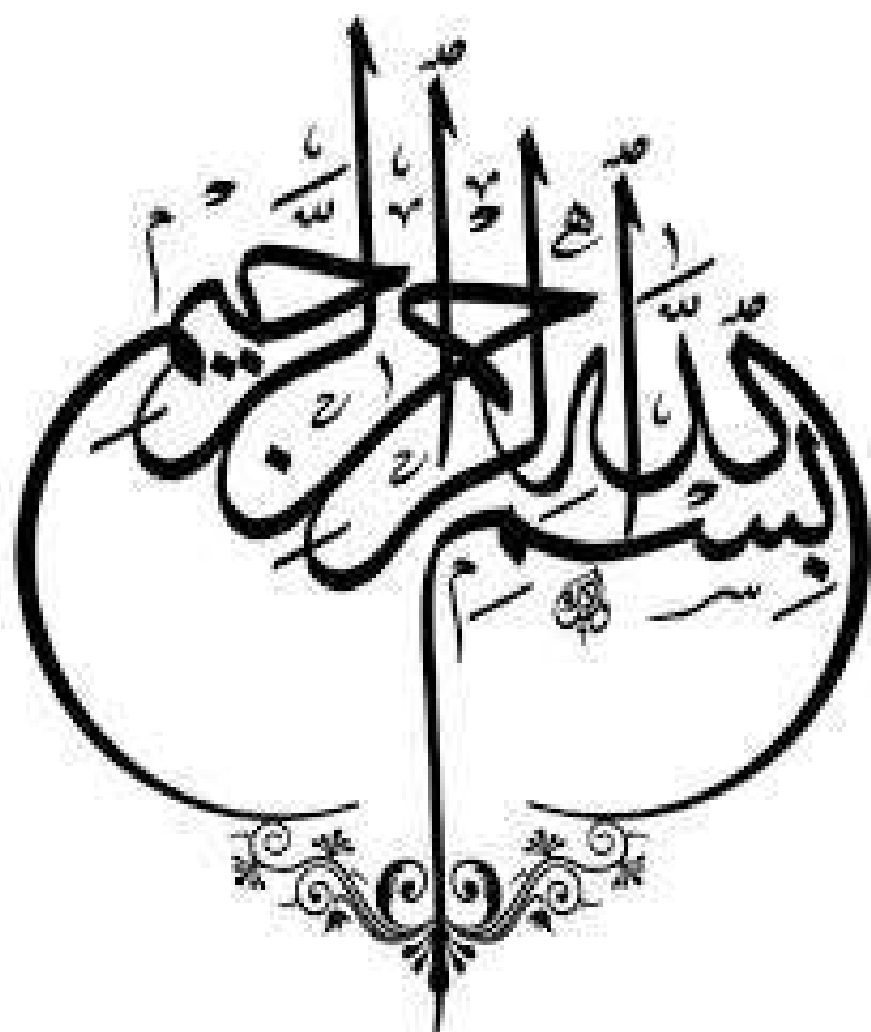
❖ **Supervisor**

Dr. Boukaous Chahra

❖ **Co -Supervisor**

Pr. Hima Abdelkader

Academic Year: 2024 / 2025



الملخص

يهدف هذا العمل إلى تصميم وتنفيذ نموذج عملي لكاميرا ثلاثية الأبعاد باستخدام المتحكم ESP32 ، قادرة على تتبع الأجسام والتعرف على الألوان في الوقت الفعلي. يعتمد المشروع على تقنيات الذكاء الاصطناعي ورؤية الكمبيوتر، مع التركيز على تطوير أنظمة بصرية منخفضة التكلفة وفعالة في بيئات محدودة.

تم استعراض المبادئ الأساسية لرؤية الكمبيوتر، وتقديم دراسة مفصلة للمتكامل الدقيق ESP32 من حيث خصائصه وبيئة تطويره. بعد ذلك، تم تصميم نموذج عملي يتضمن كاميرا ESP32-CAM والخوارزميات اللازمة للكشف عن الأجسام والتعرف على الألوان.

أظهرت النتائج أن النظام قادر على معالجة البيانات البصرية بشكل فعال، رغم القيود على الذاكرة والمعالجة. ومع ذلك، ظهرت بعض القيود في دقة تقدير العمق.

يوضح هذا البحث إمكانية تنفيذ أنظمة رؤية ثلاثية الأبعاد باستخدام منصات منخفضة التكلفة، مع اقتراح تحسينات تشمل دمج الرؤية الاستريوسكوبية واستخدام معجلات الذكاء الاصطناعي منخفضة الطاقة.

Résumé

Ce travail de recherche vise à concevoir et à réaliser un prototype fonctionnel de caméra 3D utilisant le microcontrôleur ESP32, capable de suivre des objets et de reconnaître des couleurs en temps réel. Le projet s'inscrit dans le cadre de la vision par ordinateur et de l'intelligence artificielle embarquée, visant à développer des systèmes visuels efficaces à faible coût.

Les principes de base de la vision par ordinateur ont été étudiés, ainsi qu'une analyse détaillée du microcontrôleur ESP32 et de son environnement de développement. Un prototype fonctionnel a été conçu avec la caméra ESP32-CAM et les algorithmes nécessaires.

Les résultats montrent que le système peut traiter des données visuelles en temps réel malgré les contraintes matérielles. Toutefois, des limites ont été observées dans la précision de la détection de la profondeur.

Cette recherche prouve la faisabilité de systèmes de vision 3D à faible coût, tout en suggérant des améliorations possibles, telles que l'intégration de la vision stéréoscopique et l'utilisation d'accélérateurs d'IA à faible consommation.

Summary

This research aims to design and implement a functional 3D camera prototype using the ESP32 microcontroller, capable of real-time object tracking and color recognition. The project focuses on low-cost, efficient visual systems within embedded computer vision and artificial intelligence.

The study covers the fundamentals of computer vision and a detailed review of the ESP32 microcontroller and its development environment. A functional prototype was designed using the ESP32-CAM camera and necessary algorithms.

Results show that the system can process visual data in real-time despite hardware limitations. However, depth detection accuracy showed some constraints.

This work demonstrates the feasibility of low-cost 3D vision systems and suggests improvements, such as integrating stereoscopic vision and using low-power AI accelerators.

Dedication

All praise is due to Allah, who enabled us to complete this work. We ask Him to accept it sincerely for His sake and to guide us to that which pleases Him and earns His satisfaction.

With deep humility, we dedicate the fruit of our efforts to:

The most precious person in our lives, to whom words cannot do justice - the embodiment of love and a sea of tenderness, the fragrance and joy of life, the one who gave us life and has always illuminated our path with her prayers - our beloved mother. May Allah bless her with continued health and protection.

To the greatest man in the world, who worked tirelessly for our sake, who taught us the meaning of perseverance and sacrifice, who shaped us and guided us, and who remains our greatest role model - our dear father.

To the pillars we rely on to stand strong in life, to those who raised us and surrounded us with care - our supportive brothers and sisters.

And to all our companions along the way, our fellow students and colleagues throughout our academic journey - those remembered by the heart even if not named in this document

Bennour Brahīm



Dedication

*To the Master and Teacher of humanity, the beloved Prophet
Muhammad (peace and blessings be upon him).*

*To the essence of my being, the source of tenderness, the most precious
person*

*To the one beneath whose feet Paradise lies, the one whom the Most
Merciful has commanded me to honor*

My beloved mother.

To the one who loved, nurtured, guided, and raised me

*To the one who stayed awake through the nights and toiled through
the days*

My dear father.

To the siblings of my soul, the bridge of love and loyalty

My brothers and sisters.

*To those who enlightened my ignorance, nurtured my spirit, and
guided my path*

My esteemed professors.

*And to everyone who helped me, brought joy to my life, remained in
my heart though forgotten by my pen I dedicate this work.*

Melik Brahim



Acknowledgments

All praise is due to Allah, whom we seek for help and thank for His guidance. He has made our path easier and has helped us overcome the difficulties, allowing us to complete this humble work.

We extend our sincere thanks and deepest appreciation to all the professors who accompanied us throughout this journey. We are especially grateful to our supervising professor, **Dr. Boukaous Chahra & Dr. Hima Abdelkader** for her valuable guidance, continuous support, and generosity throughout the completion of this dissertation. May Allah reward her with the best.

Our heartfelt thanks also go to the honorable members of the jury:

- **Pr. Bessous Noureddine**, President of the Committee
- **Dr. Labiod Chouaib**, Examiner
- **Dr. Boukaous Chahra**, Supervisor
- **Dr. Hima Abdelkader**, Assistant Professor

We also express our gratitude to all the staff of the Department and to the thesis committee for accepting to evaluate, discuss, and enrich this work with their insights.

Special thanks to everyone who supported us—near or far—even with a single word. May Allah reward you all abundantly.

Finally, our deepest gratitude goes to everyone who taught us, helped us, or prayed for us in silence.

Thank you all.



Table of Contents

Summary	I
Dedication	III
Dedication	IV
Acknowledgments	V
Table of contents	VI
List of tables	X
List of figures	XI
List of abbreviations	XIII
General introduction	1
Chapter I: Artificial intelligence in stabilization and tracking systems	1
<i>I.1 Introduction</i>	2
<i>I.2 Definition of artificial intelligence</i>	2
<i>I.3 Categories of AI</i>	3
<i>I.3.1 Based on capability</i>	3
<i>I.3.2 Based on functionality</i>	3
<i>I.3.3 Intelligence in machines vs. humans</i>	3
<i>I.4 Historical development of artificial intelligence</i>	4
<i>I.5 Key technologies in artificial intelligence</i>	5
<i>I.5.1 Machine learning (ML)</i>	5
<i>I.5.2 Deep learning (DL)</i>	6
<i>I.5.3 Natural language processing (NLP)</i>	7
<i>I.5.4 Computer vision</i>	7
<i>I.5.5 Robotics and intelligent agents</i>	7
<i>I.5.6 Expert systems</i>	8
<i>I.5.7 Knowledge representation and reasoning (KRR)</i>	8
<i>I.6 Benefits and opportunities of artificial intelligence</i>	9
<i>I.7 Challenges and ethical concerns in artificial intelligence</i>	10
<i>I.8 The future of artificial intelligence</i>	11
<i>I.9 Application of artificial intelligence</i>	13
<i>I.9.1 Healthcare</i>	13
<i>I.9.2 Finance</i>	13
<i>I.9.3 Manufacturing and industry</i>	14
<i>I.9.4 Transportation</i>	14
<i>I.9.5 Education</i>	14

I.9.6 Agriculture	14
I.9.7 Cyber security	14
I.10 Application of artificial intelligence in stabilization	14
I.10.1 Predictive analytics for early intervention	15
I.10.2 Real-time response and adaptability	15
I.10.3 Sector-specific applications	15
I.11 Comparative of general and stabilization-oriented applications of artificial intelligence	16
I.12 Challenges and considerations	17
I.13 Conclusion	17
Chapter II: Embedded systems and ESP32 microcontroller	18
II.1 Introduction	18
II.2 Embedded systems	18
II.2.1 Definition	18
II.2.2 Characteristics of embedded systems	18
II.2.3 General architecture	19
II.2.4 Classification of embedded systems based on application domains	20
II.2.4.1 General-purpose embedded systems	20
II.2.4.2 Real-time control systems	20
II.2.4.3 Digital signal processing (DSP) systems	20
II.2.4.4 Communication and networking systems	20
II.3 Microcontrollers	20
II.3.1 Definition	20
II.3.2 Advantages	22
II.3.3 Memories	22
II.3.4 Choice of a microcontroller	24
II.3.4.1 Key selection criteria	24
II.3.4.2 Popular microcontroller families	25
II.3.4.3 Selected microcontroller for this project	25
II.4 The ESP32 microcontroller	25
II.4.1 Resources and features of the ESP32	25
II.4.2 Pin configuration and GPIO considerations	26
II.5 Presentation of the ESP32-CAM board	28
II.5.1 ESP32-CAM features	28
II.5.2 Pinout of the ESP32-CAM board	29

<i>II.6 Study of the ESP32 library functions (I/O, Wi-Fi)</i>	29
<i>II.6.1 Input/output functions (GPIO)</i>	29
<i>II.6.2 I/O port libraries</i>	29
<i>II.6.2.1 Pin mode</i>	29
<i>II.6.2.2 Digital write</i>	30
<i>II.6.2.3 Digital read</i>	30
<i>II.6.2.4 Attach interrupt</i>	30
<i>II.6.2.5 Attach interrupt arg</i>	30
<i>II.6.2.6 Detach interrupt</i>	30
<i>II.6.3 Wi-Fi API</i>	30
<i>II.6.3.1 Access point (AP) mode</i>	31
<i>II.6.3.2 Operation in station (STA) mode</i>	31
<i>II.6.4 Examples of Wi-Fi functions</i>	32
<i>II.6.5 ESP_CAMERA library</i>	32
<i>II.7 Conclusion</i>	33
 Chapter III: Hardware implementation of the 3D vision system	34
<i>III.1 Introduction</i>	34
<i>III.2 Practical implementation</i>	34
<i>III.2.1 Hardware components</i>	34
<i>III.2.2 Electrical circuit design</i>	38
<i>III.2.2.1 ESP32-CAM pan-tilt system wiring diagram</i>	38
<i>III.2.2.2 USB-to-TTL serial communication interface</i>	40
<i>III.2.2.3 Servo motor connections</i>	42
<i>III.3 Software part</i>	44
<i>III.3.1 Integrated object tracking system using PC client&\$ ESP32-CAM</i>	44
<i>III.3.1.1 PC client side – object tracking and command sending</i>	44
<i>III.3.1.2 ESP32-CAM server side – command reception and execution</i>	48
<i>III.3.2 Explanation of key functions in the system</i>	49
<i>III.3.2.1 First flowchart – PC client side</i>	50
<i>III.3.2.2 Second flowchart – ESP32-CAM server side</i>	50
<i>III.4 System results and dynamic operation</i>	52
<i>III.4.1 A final image of the project</i>	52
<i>III.4.2 Electrical components of a circuit</i>	53
<i>III.4.3 System operation and experimental validation</i>	54

III.4.3.1 Before detection	55
III.4.3.2 After detection	55
III.4.4 Steps of target detection and tracking process	56
III.4.4.1 Step 1: Initial search mode (scanning phase)	57
III.4.4.2 Step 2: Object detection	57
III.4.4.3 Step 3: Target acquisition and camera adjustment	57
III.4.4.4 Step 4: Tracking confirmation (visual feedback)	57
III.5 Conclusion	58
General conclusion	59
References	60
Appendices	65

List of Tables

Chapter I: Artificial Intelligence in Stabilization and Tracking Systems	
Table name	Page number
Table I.1: Stabilization-Oriented Applications Vs Artificial Intelligence	16
Chapter 02: Embedded Systems and ESP32 Microcontroller	
Table name	Page number
Table II.1: Functional Units of an Embedded System and Their Roles	19
Table II.2: Memory Types in Embedded Systems and Processor Architecture Overview	23
Table II.3: Hardware Resources and Functional Features of the ESP32	27
Table II.4: Summary of ESP32's Built-in Resources and Capabilities	28
Chapter 03: Hardware Implementation of the 3D Vision System	
Table name	Page number
Table III.1: System Components and Their Roles in the ESP32-CAM Based Object Tracking Setup.	54

List of figures

Chapter I: Artificial Intelligence in Stabilization and Tracking Systems	
Figure name	Page number
Figure I.1: Using data to create smart technology.	2
Figure I.2: Compares machine learning and deep learning.	6
Figure I.3: Different areas of AI connect.	9
Figure I.4: Ai Applications.	13
Chapter 02: Embedded Systems and ESP32 Microcontroller	
Figure name	Page number
Figure II.1: General Structure of an Embedded System	19
Figure II.2: Internal Components of a Microcontroller.	21
Figure II.3: Types of Memories.	22
Figure II.4: Examples of Microcontroller Models.	24
Figure II.5: ESP32 Resources.	26
Figure II.6: ESP32 Microcontroller Pinout.	27
Figure II.7: ESP32-CAM Board.	28
Figure II.8: ESP32-CAM Pinout.	29
Figure II.9: Access Point Mode Operation.	31
Figure II.10: Operation in Station Mode.	31
Chapter 03: Hardware Implementation of the 3D Vision System	
Figure name	Page number
Figure III.1: ESP32-CAM.	35
Figure III.2: MG90S Micro Servo Motor.	35

Figure III.3: Micro SD Card.	36
Figure III.4: USB to TTL Serial Converter.	36
Figure III.5: Jumper Wires.	37
Figure III.6: Pan-Tilt Mechanism with Dual SG90 Micro Servo Motors.	37
Figure III.7: power supply.	38
Figure III.8: Wiring Diagram of the ESP32-CAM Pan-Tilt Object Tracking System.	39
Figure III.9: ESP32-CAM Programming and Serial Communication Interface.	41
Figure III.10: A diagram showing how to connect the esp32cam in 2 Servo Motor.	42
Figure III.11: ESP32-CAM (Flowchart: Object Tracking Program using ESP32-CAM).	45
Figure III.12: System Interface During Connection Attempt.	46
Figure III.13: User Interface with Color Selection After Successful Connection.	46
Figure III.14: ESP32-CAM Server Side – Command Reception and Camera Control.	48
Figure III.15: 3D Camera Stabilization and Tracking System with ESP32 (Pan-Tilt 180° Support)	52
Figure III.16: Final form of the project.	53
Figure III.17: Circuit installation.	54
Figure III.18: (a) System State Before Object Detection , (b) Camera Scanning Without Identifying the Target.	55
Figure III.19: (a) System State After Detecting a Blue Object , (b) Camera Locked on Target and Tracking Initialized.	56
Figure III.20: Target Detection and Tracking Process.	57

List of abbreviations

Abbreviation	Full Form
ADC	Analog-to-Digital Converter
AGI	Artificial General Intelligence
AI	Artificial Intelligence
API	Application Programming Interface
ASIC	Application-Specific Integrated Circuit
AWG	American Wire Gauge
BGR	Blue, Green, Red (Used in OpenCV)
BLE	Bluetooth Low Energy
CAN	Controller Area Network
CH340G	CH340G Serial Converter
CNN	Convolutional Neural Network
CP2102	Silicon Labs CP2102 (USB-to-Serial Converter)
CV	Computer Vision
DAC	Digital-to-Analog Converter
DL	Deep Learning
DRAM	Dynamic Random-Access Memory
EEPROM	Electrically Erasable Programmable Read-Only Memory
EPROM	Erasable Programmable Read-Only Memory
ESP	Espressif Systems Processor
ESP32-CAM	ESP32 Camera Module
FTDI	Future Technology Devices International
FS	File System
GPIO	General Purpose Input/Output
GPT	Generative Pre-trained Transformer
GND	Ground
GUI	Graphical User Interface
HMI	Human-Machine Interface
HTTP	HyperText Transfer Protocol
HSV	Hue, Saturation, Value (Color Space)
I ² C	Inter-Integrated Circuit
I ² S	Integrated Interchip Sound
IDE	Integrated Development Environment

INO	Arduino Sketch File Format
IoT	Internet of Things
IRQ	Interrupt Request
ISP	In-System Programming
KRR	Knowledge Representation and Reasoning
LED	Light Emitting Diode
MCPWM	Motor Control Pulse Width Modulation
ML	Machine Learning
NLP	Natural Language Processing
OpenCV	Open Source Computer Vision Library
OV2640	OmniVision OV2640 (2MP Camera Sensor)
PDAs	Personal Digital Assistants
PROM	Programmable Read-Only Memory
PSRAM	Pseudo Static Random-Access Memory
PWM	Pulse Width Modulation
RAM	Random Access Memory
RGB	Red, Green, Blue (Color Model)
RISC	Reduced Instruction Set Computing
RNN	Recurrent Neural Network
ROM	Read-Only Memory
RTC	Real-Time Clock
RTOS	Real-Time Operating System
SD	Secure Digital
SD_MMC	Secure Digital MultiMediaCard Library
SoC	System on Chip
SPI	Serial Peripheral Interface
SRAM	Static Random-Access Memory
STA	Station (Wi-Fi Station Mode)
TX / RX	Transmit / Receive
UART	Universal Asynchronous Receiver-Transmitter
USB	Universal Serial Bus
UXGA	Ultra Extended Graphics Array
VGA / SVGA	Video Graphics Array / Super VGA
Wi-Fi	Wireless Fidelity
WPA	Wi-Fi Protected Access
WPA2	Wi-Fi Protected Access II
WPS	Wi-Fi Protected Setup
XAI	Explainable Artificial Intelligence



General introduction

General introduction

The rapid evolution of intelligent technologies spanning smart surveillance, autonomous navigation, and human-computer interaction has underscored the transformative power of Artificial Intelligence (AI) [6]. Within embedded systems, AI plays a pivotal role in enabling devices to perceive, decide, and act with increasing autonomy and precision. Camera-based stabilization and tracking systems, in particular, have benefited immensely from AI techniques, which allow for real-time object recognition, motion prediction, and adaptive control.

This dissertation investigates the design and implementation of a 3D camera stabilization and tracking system using the ESP32-CAM microcontroller [33]. The system integrates computer vision, servo control, and wireless communication, forming a compact and intelligent solution for object tracking in dynamic environments. Key aspects include the use of AI-driven algorithms for object detection and motion tracking, as well as embedded system design principles to ensure efficiency and responsiveness.

The work aims to bridge the gap between theoretical AI concepts and practical deployment in low-cost embedded platforms. Through the integration of hardware and software components, the project seeks to demonstrate how real-time image processing and mechanical actuation can be harmonized to deliver reliable stabilization and tracking performance.

The dissertation is structured as follows:

Chapter I introduces the foundational concepts of Artificial Intelligence [6], its classifications, historical evolution, and core techniques such as machine learning, deep learning, computer vision, and natural language processing. It also explores the ethical and societal implications of AI, as well as its applications in stabilization systems.

Chapter II explores the principles of embedded systems, microcontroller architectures, and the ESP32 platform [33]. It discusses the components relevant to the project including ESP32-CAM, servos, and peripheral modules and provides an overview of camera interfacing and communication protocols.

Chapter III describes the design and implementation of the proposed system, presenting the hardware setup, control algorithms, and graphical interface [28]. It includes the development environment, software libraries used, and detailed explanations of object detection, image streaming, and servo coordination.

The dissertation concludes by highlighting the practical outcomes of the system, evaluating its performance, and suggesting potential improvements and future directions in AI-powered embedded tracking solutions.

A rectangular area with a marbled, stone-like texture in shades of gray. It has a black border and rounded corners on the left and right sides.

Chapter I: Artificial Intelligence in Stabilization and Tracking Systems

I.1 Introduction

Artificial Intelligence (AI) has become one of the most transformative technologies in modern engineering and computing. It plays a pivotal role in enabling machines to simulate human intelligence, such as learning, reasoning, perception, and decision-making. This chapter aims to introduce the foundational concepts of AI, differentiate it from human intelligence, and explore its evolution over time.

We begin by defining AI and presenting its main types, including narrow AI, general AI, and superintelligence. The chapter then explores the differences between artificial and human intelligence, followed by a historical overview of AI development. Furthermore, we highlight core AI technologies such as machine learning, deep learning, natural language processing, computer vision, and robotics. Particular focus is placed on real-world applications of AI across various sectors, especially in the fields of stabilization and tracking systems. The chapter concludes with a discussion on the ethical challenges and future directions of AI, laying the foundation for understanding how AI contributes to the development of intelligent 3D camera stabilization and tracking systems, which are the focus of this dissertation.

I.2 Definition of Artificial Intelligence

Artificial Intelligence (AI) is a foundational concept in computer science that refers to the ability of machines to perform tasks that typically require human intelligence. These tasks include learning from experience, reasoning, understanding language, perceiving the environment, and making decisions. The formal study of AI spans multiple disciplines such as mathematics, cognitive science, linguistics, and engineering, and its applications are increasingly embedded in daily life [1].

This figure I.1 illustrates how data serves as the foundation for building intelligent systems by training algorithms that enable machines to make decisions or predictions.

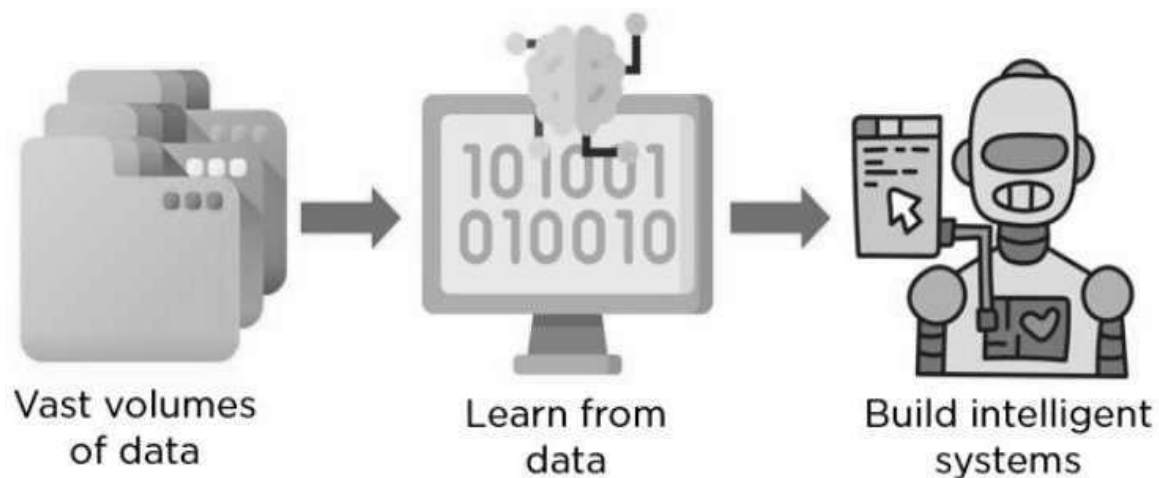


Figure I.1: Using data to create smart technology [2].

- **What Is AI?**

One of the earliest and most widely cited definitions of AI was provided by John McCarthy, a founding father of the field, who stated:

Artificial Intelligence is the science and engineering of making intelligent machines, especially intelligent computer programs.

Building on this, Russell and Norvig (2021) categorize AI as:

The study of agents that receive percepts from the environment and perform actions.”

Artificial Intelligence: A Modern Approach

These definitions emphasize the creation of systems that can simulate human-like thinking and behavior. However, modern AI also includes subfields that seek to replicate specific cognitive functions, rather than general intelligence [3].

I.3 Categories of AI

AI can be classified in multiple ways based on capability and functionality:

I.3.1 Based on Capability

- **Narrow AI (Weak AI):** Designed to perform a single task (e.g., voice assistants like Alexa, facial recognition software). This is the dominant form of AI today.
- **General AI (Strong AI):** A theoretical system with human-level cognitive abilities, capable of understanding, reasoning, and learning across different tasks.
- **Super intelligent AI:** A hypothetical form of AI that surpasses human intelligence in all aspects reasoning, creativity, and decision-making.

I.3.2 Based on Functionality

- **Reactive Machines:** Systems with no memory, reacting only to current inputs (e.g., IBM’s Deep Blue chess computer).
- **Limited Memory AI:** Systems that learn from historical data (e.g., self-driving cars).
- **Theory of Mind AI:** Still in development; aims to understand emotions and mental states.
- **Self-Aware AI:** A speculative future system that possesses consciousness and self-awareness.

I.3.3 Intelligence in Machines vs. Humans

While human intelligence is adaptive, emotional, and general-purpose, AI is primarily task-specific and depends on data and algorithms. However, with advancements in deep learning and natural language processing, machines are increasingly able to:

- Interpret language and emotions
- Learn from unstructured data

- Make real-time decisions in dynamic environments

This evolution is blurring the boundary between programmed automation and adaptive intelligence [4].

1.4 Historical Development of Artificial Intelligence

The story of Artificial Intelligence is a rich tapestry woven through centuries of philosophical curiosity, scientific breakthroughs, and technological ambition. Long before the term "AI" existed, humanity dreamed of imbuing machines with intelligence. Ancient myths from Greece spoke of intelligent automatons created by gods, while early mechanical devices in China hinted at our fascination with life-like machines.

Yet it wasn't until the 17th and 18th centuries that this fascination began to take a structured form. Thinkers like René Descartes and Leibniz pondered whether reasoning could be mechanized. Then, in 1837, Charles Babbage designed the Analytical Engine — a machine capable of processing instructions. Ada Lovelace, in a visionary leap, suggested it might one day be used for something beyond numbers: symbolic reasoning [5].

The 20th century laid the true mathematical and theoretical foundation of AI. In 1936, Alan Turing introduced the idea of a “universal machine.

- what we now call a Turing Machine
- a model that defined computation in abstract terms. Turing's legacy continued in 1950 with the famous “Turing Test,” asking a bold question: Can machines think?
- Then came 1956, a year widely regarded as the birth of artificial intelligence as a formal discipline. At Dartmouth College, a small group of scientists
- including John McCarthy, Marvin Minsky, and Claude Shannon
- gathered to define a new research field. It was McCarthy who coined the term “Artificial Intelligence,” marking the beginning of decades of exploration, hope, and challenge.

The early years were filled with excitement. Basic AI programs like ELIZA (1966), which simulated conversation, and SHRDLU (1970), which could manipulate virtual objects, suggested early promise. However, by the 1970s, the reality did not meet the lofty expectations. Funding slowed, interest waned, and the field entered what became known as the First AI Winter [6].

The 1980s saw a resurgence with the rise of expert systems programs designed to mimic the decision-making abilities of human experts. Systems like XCON helped companies solve real-world problems, especially in configuration and diagnostics. But these systems lacked flexibility. They worked well only in narrow domains. Eventually, their limitations led to the Second AI Winter by the end of the decade.

Still, research persisted. In 1997, a monumental moment occurred: IBM's Deep Blue defeated world chess champion Garry Kasparov. It was the first time a machine had triumphed over a human in a cognitively complex domain, sending a clear signal that AI was far from dead.

By the 2000s, statistical learning, neural networks, and access to vast data transformed the AI landscape. And then, in 2012, something extraordinary happened. A neural network called Alex Net outperformed all competitors in the Image Net competition, marking the start of the deep learning revolution. Using large datasets and GPU computing, machines could now recognize images, understand speech, and translate text with unprecedented accuracy.

The pace quickened. In 2016, Alpha Go, developed by Deep Mind, defeated the world champion of Go a game long considered too complex for machines. By 2020, AI was writing coherent essays, composing music, and driving cars.

Today, AI is embedded in every part of life from voice assistants and medical diagnostics to financial modeling and climate prediction. Yet, despite its immense capabilities, current AI remains narrow: powerful in specific tasks but lacking human-like general reasoning. The quest for Artificial General Intelligence (AGI) an AI that can learn any task continues.

As we look ahead, the focus is no longer just on what AI can do, but what it should do. Ethical AI, transparency, and social responsibility are now central to the conversation. The path from myth to machine has been long, and the journey is far from over [7].

I.5 Key Technologies in Artificial Intelligence

Artificial Intelligence (AI) is a field composed of multiple core technologies, each of which plays a critical role in enabling machines to mimic or enhance aspects of human cognition. These technologies span from learning algorithms and symbolic reasoning to perception and physical interaction. Below is a detailed explanation of the foundational technologies that power today's intelligent systems.

I.5.1 Machine Learning (ML)

Machine Learning is the cornerstone of AI. It enables systems to automatically learn from data, recognize patterns, and improve performance over time. Rather than relying on hard-coded instructions, ML models use statistical techniques to make predictions or decisions based on historical data.

Supervised Learning involves training algorithms on labeled datasets, where inputs are paired with correct outputs. It is widely used in tasks such as spam detection, medical diagnosis, and customer churn prediction.

Unsupervised Learning works with unlabeled data, aiming to uncover hidden patterns or groupings. This is useful in customer segmentation, anomaly detection, and data compression.

Reinforcement Learning models learn through interaction with an environment, using rewards and punishments to shape behavior. It is commonly applied in robotics, gaming (e.g., Alpha Go), and autonomous navigation systems.

These approaches collectively enable machines to generalize from experience, a defining trait of intelligent behavior [8].

1.5.2 Deep Learning (DL)

Deep Learning, a specialized subfield of ML, uses artificial neural networks with multiple layers (hence “deep”) to model complex relationships in data. These architectures are inspired by the structure and function of the human brain, particularly the neocortex [9].

This figure I.2 highlights the differences between machine learning and deep learning in terms of data processing, architecture, and required human intervention.

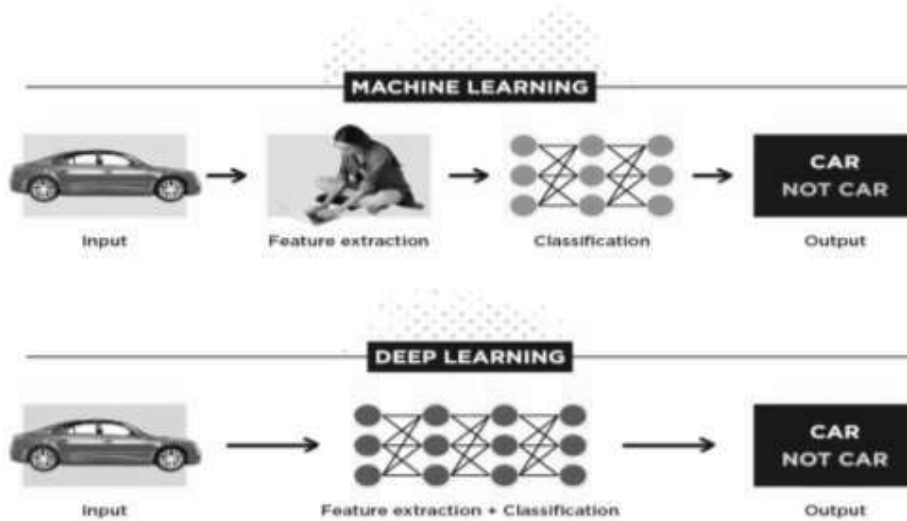


Figure I.2: Compares machine learning and deep learning [10].

Deep learning models excel at processing high-dimensional inputs such as:

- Images (via convolutional neural networks - CNNs)
- Text (via recurrent neural networks - RNNs and transformers)
- Audio and video streams

They have revolutionized fields like computer vision, speech recognition, and language modeling by achieving human-comparable or even superior accuracy in some domains. Despite their power, DL models are computationally intensive and often considered “black boxes,” making them difficult to interpret.

I.5.3 Natural Language Processing (NLP)

NLP is the branch of AI that enables machines to understand, generate, and respond to human language. It combines linguistics, computer science, and statistics to bridge the gap between human communication and machine processing. And the key subfields and applications include:

- Syntax and parsing for grammatical structure analysis
- Named entity recognition (e.g., identifying people, locations in text)
- Machine Translation between languages
- Question answering and conversational agents (e.g., Siri, Chat GPT)
- Large language models like GPT-4 and BERT that use billions of parameters to generate coherent and contextually relevant language.

Recent innovations in transformer architectures have dramatically improved performance in NLP, enabling machines to understand context and nuance in ways that were previously unattainable.

I.5.4 Computer Vision

Computer Vision enables machines to analyze and interpret visual information from the world. It seeks to replicate and extend the human ability to see and understand the visual environment. Core applications include:

- Image classification (e.g., identifying diseases from X-rays)
- Object detection (e.g., recognizing pedestrians for self-driving cars)
- Facial recognition (e.g., security and identity verification)
- Scene segmentation for understanding spatial layouts in images
- Augmented reality and 3D modeling

These tasks often rely on CNNs, which learn to extract visual features hierarchically from edges and textures to complex object structures.

I.5.5 Robotics and Intelligent Agents

Robotics integrates AI with physical machines to perform actions in the real world.

- **Intelligent robots combine:**
 - ✓ Sensors to perceive their environment
 - ✓ Actuators to perform movements
 - ✓ Planning algorithms to decide on actions
 - ✓ Learning algorithms to adapt to new situations
- **Applications include:**
 - ✓ Industrial automation (e.g., robotic arms in manufacturing)

- ✓ Healthcare robots (e.g., surgery assistants, elderly care)
- ✓ Exploration (e.g., planetary rovers, underwater drones)

Modern robotic systems often incorporate reinforcement learning and computer vision to navigate dynamic environments with limited human supervision.

I.5.6 Expert Systems

Expert systems were among the earliest forms of AI. They are designed to replicate the decision-making ability of a human expert in specific domains.

- **Components include:**
 - ✓ Knowledge Base: A collection of facts and rules about the domain
 - ✓ Inference Engine: Logical framework that applies rules to known facts to infer new knowledge
 - ✓ User Interface: Allows users to interact with the system and receive decisions or advice
- **Use cases include:**
 - ✓ Medical diagnostics (e.g., MYCIN, a system for infectious diseases)
 - ✓ Financial advising systems
 - ✓ Legal decision support

While modern AI often favors data-driven approaches, expert systems remain valuable in domains where human expertise can be formally encoded and where explainability is essential [9] [10].

I.5.7 Knowledge Representation and Reasoning (KRR)

KRR is a symbolic AI method that enables machines to represent facts, relationships, and logic rules about the world. It is crucial for:

- Understanding meaning in natural language
- Building ontologies for domains like medicine or law
- Drawing conclusions from known information using formal logic (e.g., predicate logic, semantic networks)
- Supporting explainable AI systems by allowing machines to reason step-by-step

Unlike ML models, KRR systems are inherently interpretable and ideal for domains requiring high levels of transparency and trust, such as healthcare policy or legal reasoning. Each of these technologies contributes uniquely to the field of AI and formal reasoning. The future of AI lies in the integration of these technologies combining statistical learning, perception, and logic to create systems that are both powerful and accountable.

The figure I.3 shows the relationship and overlap between various subfields of artificial intelligence, such as machine learning, computer vision, and natural language processing [9] [10].

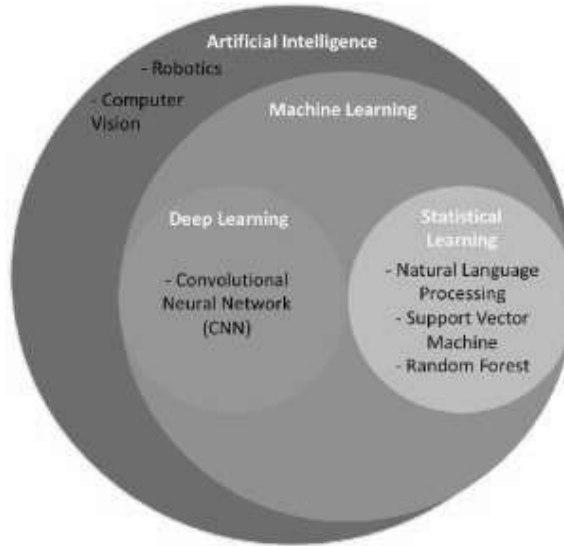


Figure I.3: Different areas of AI connect [11].

While machine learning and deep learning dominate current applications, symbolic approaches like KRR and expert systems remain essential for explain ability.

I. 6 Benefits and Opportunities of Artificial Intelligence

Artificial Intelligence (AI) is increasingly recognized as a transformative force across all dimensions of contemporary life, offering both immediate applications and long-term strategic value. One of its most profound contributions lies in the economic domain, where AI facilitates process automation, operational efficiency, and new business models. From intelligent supply chain management to real-time financial forecasting, organizations are leveraging AI to minimize costs, optimize productivity, and make more informed decisions. According to Banafa (2025), AI implementation across industries has already yielded measurable growth in productivity and market agility, particularly in sectors such as finance, logistics, and manufacturing.

Beyond economic gains, AI opens new avenues in healthcare and life sciences. Machine learning algorithms trained on large datasets have demonstrated unprecedented capabilities in medical image analysis, disease prediction, and personalized medicine. AI-driven diagnostic tools now support physicians in detecting diseases like cancer and Alzheimer's at earlier stages with higher accuracy. Furthermore, AI accelerates pharmaceutical research by identifying potential compounds and predicting molecular behavior, thereby reducing the time and cost of drug development [12].

In the context of societal advancement, AI promotes inclusivity, accessibility, and personalized experiences. Educational technologies powered by AI adapt learning materials to individual student needs, enabling more equitable access to quality education. Similarly, AI-driven language tools help bridge communication gaps for people with disabilities or those in multilingual environments. In public administration, AI supports policy-making through predictive analytics and simulation modeling, enhancing governance and service delivery.

AI also plays a critical role in environmental sustainability. It is being used to optimize energy consumption in smart grids, forecast climate patterns, monitor biodiversity, and support disaster response systems. Bhat and Infascelli (2025) emphasize that AI-enhanced environmental monitoring tools are essential for achieving the goals of the circular economy and sustainable development.

Perhaps most importantly, AI fosters scientific discovery and innovation. By rapidly analyzing vast, complex datasets, AI tools enable researchers to generate hypotheses, model scenarios, and uncover insights that would be otherwise impossible to achieve through traditional methods. In genomics, AI accelerates the mapping of genetic data; in materials science, it predicts properties of new compounds; in astrophysics, it identifies exoplanets in telescope data.

In conclusion, the benefits and opportunities of artificial intelligence are broad, multifaceted, and far-reaching. AI has already moved beyond theoretical promise into tangible, practical impact shaping how societies function, economies grow, and individuals live and learn. Its future potential depends on how responsibly, inclusively, and strategically it is developed and applied [13].

I.7 Challenges and Ethical Concerns in Artificial Intelligence

As Artificial Intelligence (AI) systems grow in complexity and scope, the ethical and societal implications of their use have become a critical area of concern. The discourse around AI today is no longer limited to performance and efficiency but extends deeply into questions of accountability, fairness, transparency, and the preservation of fundamental human rights.

A major ethical issue stems from algorithmic bias. AI systems, especially those trained on large-scale datasets, may inherit and even amplify societal biases present in the data. Studies such as those by Buolamwini and Gebru (2018) demonstrated racial and gender disparities in facial recognition systems, raising concerns about discrimination in law enforcement and public surveillance. Despite advances in fairness-aware machine learning, ensuring unbiased outcomes remains a technically and socially unresolved challenge.

Closely related is the problem of lack of transparency or what is often termed the "black box" nature of AI. Deep learning models, while highly accurate, provide little explanation for

their decisions, making them unsuitable for contexts where interpretability is essential — such as in healthcare diagnoses or judicial decision-making. The emerging field of Explainable AI (XAI) aims to address this issue, but there remains a trade-off between interpretability and model complexity.

Another major challenge lies in data privacy and surveillance. AI systems are often trained on vast amounts of personal data, raising critical questions about consent, data ownership, and individual rights. The use of AI in behavioral prediction, targeted advertising, and biometric surveillance poses risks of intrusive monitoring, especially when implemented by authoritarian regimes or without adequate regulation.

The challenge is exacerbated in the context of generative AI, where models can synthesize content that closely resembles real individuals, blurring the line between synthetic data and identity manipulation.

Furthermore, job displacement and economic disruption present a socio-economic concern. Automation powered by AI threatens to replace not only manual labor but also cognitive tasks, affecting sectors like transportation, customer service, journalism, and even education. While some economists argue that AI will create new types of jobs and boost productivity, others warn of growing inequality and social instability if appropriate retraining programs and safety nets are not established.

Perhaps the most complex concern is the problem of moral and legal responsibility. When AI systems make mistakes in areas like autonomous driving or automated financial trading it remains unclear who should be held accountable: the developer, the user, or the system itself. This legal gray area complicates the integration of AI into high-stakes applications and calls for the development of robust governance frameworks and international regulation.

Finally, the long-term debate over autonomous weapons and super intelligent AI raises existential risks. The idea that machines could one day operate beyond human control is no longer purely science fiction. Leading AI researchers, including those at Open AI and the Future of Life Institute, have advocated for global agreements to prevent AI from being weaponized or evolving without alignment to human values [14] [15].

1.8 The Future of Artificial Intelligence

The trajectory of Artificial Intelligence (AI) is no longer speculative it is unfolding rapidly across global systems. As AI continues to mature, its future holds transformative potential alongside unprecedented ethical, economic, and geopolitical challenges. Forecasts suggest that AI will not only redefine how machines function but also how humans interact with knowledge, labor, and each other.

In the next decade, AI is expected to move beyond task-specific systems to more generalized intelligence, capable of adapting to unfamiliar problems across diverse domains. Research into Artificial General Intelligence (AGI) systems with human-comparable cognitive flexibility is progressing steadily, with companies such as Open AI, Deep Mind, and Anthropic investing heavily in scalable architectures and reinforcement learning models. While AGI remains theoretical for now, its future realization poses profound questions regarding alignment with human values, oversight, and control mechanisms.

Simultaneously, AI will increasingly become embedded in the infrastructure of daily life. Smart cities, autonomous transport, and ambient intelligent environments will become more interconnected through AI-driven Internet of Things (IoT) systems. AI is also expected to accelerate innovation in personalized medicine, synthetic biology, climate modeling, and space exploration domains where data complexity exceeds human analytic capacity.

However, with increased capability comes heightened risk. Experts warn of scenarios where AI is used to manipulate information ecosystems, intensify surveillance, or automate warfare. In response, global regulatory initiatives are emerging, such as the European Union's AI Act (2021), which seeks to define risk-based classifications and transparency requirements for AI systems. UNESCO's 2021 Recommendation on the Ethics of Artificial Intelligence also emphasizes fairness, sustainability, and accountability as guiding principles for the future of AI.

Another key dimension of the AI future is its impact on the global workforce. Studies by McKinsey and the World Economic Forum indicate that while AI will automate 30–40% of existing tasks in some industries, it will also create new roles requiring hybrid human-AI collaboration. The rise of “centaur intelligence” where humans and machines co-create is a likely paradigm, necessitating large-scale investment in digital education, reskilling, and inclusive access to AI tools.

Importantly, the long-term future of AI will be shaped not just by technical breakthroughs, but by societal choices. Public trust, legal frameworks, cultural adaptation, and interdisciplinary cooperation will determine whether AI becomes a tool of empowerment or exclusion. Leading AI research institutions are now emphasizing alignment research ensuring that highly capable AI systems remain interpretable, corrigible, and beneficial to human well-being.

In conclusion, the future of Artificial Intelligence holds extraordinary promise, but demands caution, collaboration, and vision. It offers humanity the tools to solve complex problems from pandemics to planetary sustainability but also challenges our assumptions about

autonomy, responsibility, and intelligence itself. Navigating this future responsibly will require an ongoing dialogue between science, policy, ethics, and public interest [16].

I.9. Application of Artificial Intelligence

Artificial Intelligence (AI) has transcended theoretical research to become a driving force across nearly every sector of modern society. Its applications are diverse, dynamic, and continuously evolving as new algorithms and computing capabilities emerge. From healthcare to finance, and from manufacturing to education, AI is reshaping the way tasks are performed, decisions are made, and services are delivered.

This figure I.4 presents a range of real-world applications of AI in fields such as healthcare, robotics, finance, and autonomous systems.

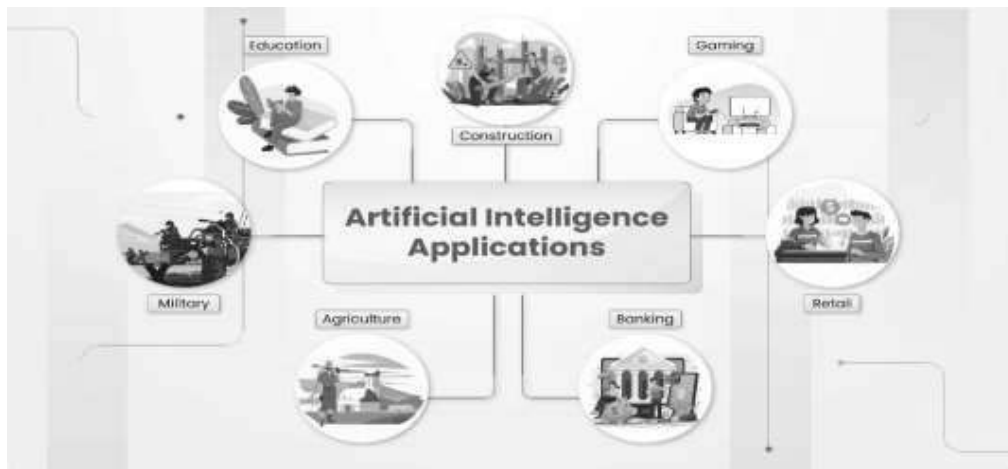


Figure I.4: AI Applications [17].

I.9.1 Healthcare

AI systems are revolutionizing healthcare through advanced diagnostics, personalized medicine, and robotic surgery. Machine learning models are used to detect diseases such as cancer from imaging scans with remarkable accuracy. Natural language processing aids in analyzing medical records, while AI-powered virtual assistants enhance patient engagement and reduce administrative burden.

Example: IBM Watson is used to support oncologists in recommending treatment options based on patient history and clinical data.

I.9.2 Finance

In the financial sector, AI is used for fraud detection, algorithmic trading, credit scoring, and customer service automation. Predictive models analyze market trends, while AI bots can execute trades faster and more accurately than human agents.

Example: AI-driven fintech platforms use real-time data to detect fraudulent activities and block suspicious transactions automatically.

I.9.3 Manufacturing and Industry

AI enables smart manufacturing through predictive maintenance, process optimization, and quality control. By analyzing sensor data, AI systems can detect equipment anomalies before failures occur, thereby reducing downtime and increasing productivity.

Example: General Electric employs AI algorithms to monitor machinery in industrial plants and optimize performance.

I.9.4 Transportation

From autonomous vehicles to intelligent traffic management systems, AI plays a central role in modernizing transportation. AI algorithms help vehicles navigate, avoid collisions, and adjust routes based on real-time traffic data.

Example: Tesla's Autopilot system integrates deep learning for self-driving capabilities, including lane keeping, adaptive cruise control, and auto-parking.

I.9.5 Education

AI tools personalize learning experiences by adapting content to individual student needs. Intelligent tutoring systems, automatic grading software, and AI chatbots enhance both teaching efficiency and student outcomes.

Example: Platforms like Coursera and Duolingo use AI to tailor content delivery, track progress, and improve learner retention.

I.9.6 Agriculture

AI helps farmers make data-driven decisions through crop monitoring, pest detection, and yield prediction. Drone imagery combined with AI algorithms improves precision farming, leading to more sustainable and productive agriculture.

Example: John Deere integrates AI in its autonomous tractors and agricultural equipment for optimized seeding and harvesting.

I.9.7 Cyber security

AI strengthens cyber security by detecting threats, preventing breaches, and managing incident response. Machine learning models identify anomalies in network behavior, signaling potential cyber-attacks.

Example: Dark trace uses AI to create digital immune systems that learn the behavior of a network and respond to cyber threats in real time [18] [19].

I.10 Application of Artificial Intelligence in Stabilization

In today's rapidly evolving world, the concept of stabilization maintaining or restoring balance in complex systems has become more critical than ever. Whether dealing with economic volatility, climate disruption, infrastructure instability, or cyber threats, stabilization

is a foundational goal across disciplines. At the heart of modern efforts to achieve and sustain such balance lies Artificial Intelligence (AI), a powerful tool with the capacity to analyze, adapt, and act in real-time.

Artificial Intelligence, by definition, refers to systems that mimic human intelligence to perform tasks such as reasoning, learning, decision-making, and pattern recognition. When applied to stabilization, AI offers capabilities that far exceed traditional static models, thanks to its ability to process vast data streams, detect subtle trends, and respond dynamically to evolving conditions.

Stabilization can take many forms, and AI's role varies depending on the domain. In engineering, it may involve maintaining the structural integrity of a bridge or building; in economics, it could mean stabilizing a volatile market; in cyber security, it refers to detecting and mitigating threats in real time. Across all these domains, AI functions not just as a monitoring tool, but increasingly as an autonomous agent capable of initiating corrective actions.

I.10.1 Predictive Analytics for Early Intervention:

One of the most valuable contributions of AI in stabilization is predictive capability. For instance, machine learning models trained on historical economic data can forecast market crashes or inflation spikes. In infrastructure systems, AI can predict structural failures by analyzing sensor data. This proactive aspect of AI allows for early intervention, often preventing damage before it occurs.

I.10.2 Real-Time Response and Adaptability:

Traditional stabilization mechanisms often rely on predefined rules and human oversight. AI, however, introduces adaptive control, particularly in environments with real-time variability. For example, in smart grids, AI adjusts energy distribution dynamically to maintain balance between supply and demand. In autonomous vehicles, AI stabilizes navigation in response to unexpected obstacles or road changes.

I.10.3 Sector-Specific Applications

a. Economic and Political Stabilization:

Governments and financial institutions use AI to track public sentiment, detect economic fraud, and stabilize currency markets. Natural Language Processing (NLP) tools analyze news and social media to anticipate unrest or misinformation spread.

b. Environmental Stabilization:

AI models predict climate trends, simulate environmental policies, and optimize water and land use in agriculture. These models are crucial in regions vulnerable to desertification or floods.

c. Structural and Engineering Stability:

In construction and transportation, AI is embedded into smart sensors and monitoring systems. These systems detect vibration, temperature changes, and stress, ensuring infrastructure safety especially in earthquake-prone zones.

d. Cyber security and Network Resilience:

AI algorithms monitor network behavior continuously, identifying potential breaches or overloads. Through anomaly detection and automated mitigation, systems remain stable even under threat [20].

I.11 Comparative of General and Stabilization-Oriented Applications of Artificial Intelligence:

The following table II.1 provides a comparative overview between general applications of Artificial Intelligence (AI) and its specialized role in stabilization contexts.

Table I.1: Stabilization-Oriented Applications Vs Artificial Intelligence.

Aspect	General Applications of AI	AI Applications in Stabilization
Purpose	Enhance productivity, automation, and decision-making across sectors	Maintain or restore balance in complex and dynamic systems
Key Domains	Healthcare, finance, manufacturing, transportation, education, agriculture, cybersecurity	Economy, environment, infrastructure, cyber networks, autonomous systems
Core Functions	Object recognition, language processing, trading automation, process optimization, virtual assistance	Predictive analytics, real-time monitoring, anomaly detection, adaptive control
Response Type	Task-specific, often static or reactive solutions	Dynamic, real-time adaptation with corrective feedback mechanisms
Use of Data	Pattern recognition from structured/unstructured data for automation	Continuous analysis of streaming data to detect deviations and prevent instability
Examples	- IBM Watson in healthcare - Tesla Autopilot - AI in Duolingo & Coursera	- AI for early detection of market crashes - Smart grids for energy balance - Structural sensors in smart buildings
Technological Tools	NLP, computer vision, CNNs, chatbots, IoT devices	Predictive ML models, smart sensors, adaptive algorithms, cyber anomaly detection
Outcome Focus	Efficiency, personalization, speed, and automation	Reliability, system safety, sustainability, and resilience

While both domains leverage AI's core abilities in learning, reasoning, and decision-making, their focus, objectives, and implementation scenarios differ significantly. The comparison

below highlights key areas including purpose, domains, functionality, and specific examples to clearly distinguish how AI adapts to diverse needs in general vs. stabilization-focused applications.

I.12 Challenges and Considerations:

Despite the advantages, the use of AI in stabilization is not without risks. Algorithmic bias, lack of transparency, and ethical dilemmas regarding autonomous decision-making are growing concerns. Over-reliance on AI could lead to unforeseen consequences, especially if systems are trained on biased or incomplete data. Moreover, in critical areas such as military operations or emergency management, human oversight remains indispensable [21].

I.13 Conclusion

Artificial Intelligence (AI) has emerged as one of the most transformative technologies of the 21st century, reshaping industries, economies, and social systems in profound ways. Through an exploration of its definition, historical development, core technologies, and diverse applications, it becomes evident that AI is not merely a computational tool but a powerful force influencing how societies function, how decisions are made, and how innovation is driven.

AI technologies including machine learning, deep learning, natural language processing, and robotics offer unparalleled opportunities for enhancing productivity, personalizing services, advancing scientific discovery, and addressing complex global challenges such as climate change and public health. At the same time, the integration of AI into critical systems raises urgent concerns around bias, transparency, privacy, accountability, and the disruption of labor markets.

The future of AI is not predetermined; it will be shaped by the collective actions of researchers, developers, policymakers, and citizens. Ethical governance, interdisciplinary collaboration, and inclusive education will be essential to ensuring that AI is developed and deployed in a manner that aligns with societal values and human dignity.

Ultimately, Artificial Intelligence holds the potential to serve as both a tool for progress and a mirror reflecting the choices we make as a global society. Its promise lies not only in its computational power but in our ability to guide it with wisdom, equity, and foresight.

Chapter II.: Embedded Systems and ESP32
Microcontroller

II.1 Introduction

Any embedded system design requires both a hardware component and a software component. An embedded system is an electronic and computer-based system driven by software, which is fully integrated into the system it controls.

The Arduino platform enables the combination of programming capabilities with those of electronics. The main advantage of programmable electronics is that it significantly simplifies electronic schematics, thereby reducing both production costs and the workload during the design phase of an electronic board.

In this chapter, we will present some fundamental concepts related to embedded systems as well as microcontrollers. We will focus particularly on the ESP32 microcontroller used in our work.

II.2 Embedded Systems

II.2.1 Definition

An embedded system is a complex autonomous system that integrates both software and hardware, designed together to deliver specific functionalities. It is often real-time, compact in size, and characterized by low power consumption.

An embedded system features an architecture similar to that of computers. It comprises a hardware component that includes a processor, various types of memory (RAM, ROM, and flash), input/output interfaces, and several ASICs. The software component consists of the program responsible for operating the electronic hardware.

II.2.2 Characteristics of Embedded Systems

Embedded systems are designed to perform specific tasks. Below is a set of common characteristics shared by the majority of embedded systems, which they are expected to fulfill.

- **Dedicated:** An embedded system is designed for a specific task or application.
- **Autonomous:** It must operate without external human intervention.
- **Real-time operation:** It must be capable of responding to events and delivering the results of operations and processing within predetermined time constraints.
- **Reliability and operational safety:** In the event of a failure, an embedded system must still be able to function correctly.
- **Low cost:** Production costs must be minimized while maintaining functionality. When manufactured in large quantities, their unit cost becomes significantly lower.
- **Compact size and weight:** Embedded systems must be portable; their volume and weight should not present an obstacle.

- Low energy consumption: Power consumption must be as low as possible, especially given that many systems rely on batteries, solar panels, or even fuel cells in certain prototypes [21].

II.2.3 General Architecture

Embedded systems generally use low-power microprocessors or microcontrollers, with the software component partially or fully programmed into the hardware typically stored in non-volatile memory such as ROM, EPROM, EEPROM, or FLASH. Figure II.1 illustrates the basic architecture of an embedded system, which may vary from one system to another.

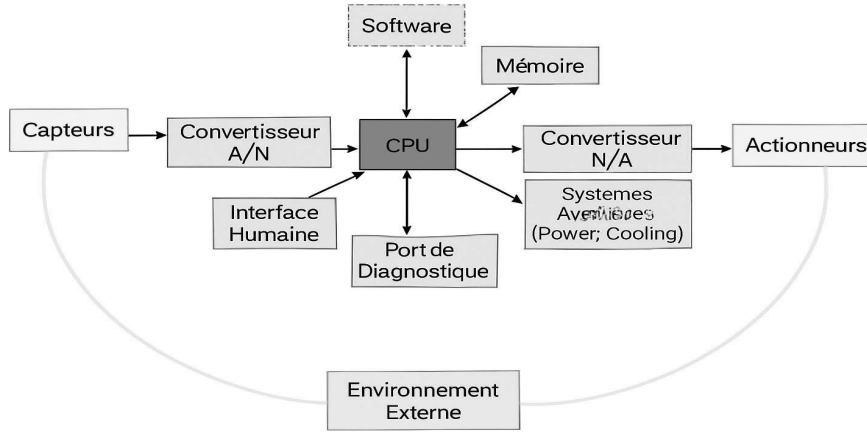


Figure II.1: General Structure of an Embedded System [22].

The architecture of an embedded system typically consists of three main functional units: the sensing unit, the processing unit, and the output unit. Each of these units plays a critical role in ensuring the system operates effectively and interacts with its environment. The following table II.1 summarizes the key components and functions of each unit:

Table II.1: Functional Units of an Embedded System and Their Roles.

Unit	Components	Function
Sensing Unit	-Sensors	Acquires data from the external environment (e.g., motion, position, light, temperature).
Processing Unit	- Processor - ADC (Analog-to-Digital Converter) - Memory (ROM, RAM, EEPROM, etc.) - Software - ASICs (Application-Specific Integrated Circuits) - DAC (Digital-to-Analog Converter)	- Processes and analyzes input data. - Converts analog signals into digital form. - Stores data and instructions. - Executes specific tasks tailored to the application. - Converts digital signals back to analog form.
Output Unit	- Actuators (e.g., motors, LEDs, displays) - HMI (Human-Machine Interface)	Acts on the environment based on processed data. Enables user interaction with the system (e.g., touchscreen, buttons).

The architecture of an embedded system can vary depending on the type of embedded system and the specific domain in which it is used. As a result, the development of embedded systems requires expertise in both electronics and computer science.

II.2.4 Classification of Embedded Systems Based on Application Domains

Embedded systems can be broadly categorized into four major types, depending on their functional objectives and target application domains:

II.2.4.1. General-Purpose Embedded Systems:

These systems are designed to execute a variety of user-oriented applications, similar to those found on desktop computers. They often support user interfaces and complex software stacks.

Examples: Personal Digital Assistants (PDAs), Automated Teller Machines (ATMs).

II.2.4.2. Real-Time Control Systems:

These systems are optimized for real-time performance, where timely and predictable responses are critical. They typically operate under the control of a Real-Time Operating System (RTOS).

Examples: Automotive engine control units, nuclear power plant monitoring, air traffic control systems.

II.2.4.3. Digital Signal Processing (DSP) Systems:

Designed specifically to process large volumes of streaming data, DSP-based embedded systems are used in applications requiring high-speed mathematical computations.

Examples: Audio/video processing devices, radar systems, sonar equipment.

II.2.4.4. Communication and Networking Systems:

These embedded platforms focus on transmitting, receiving, and managing data across networks. They often support protocols and interfaces required for communication infrastructure.

Examples: Routers, telephony systems, and Internet-of-Things (IoT) devices [23].

II.3. Microcontrollers

II.3.1 Definition

A microcontroller is an integrated circuit that combines, within a single chip, the essential elements of a computer: processor, memory (both ROM and RAM), peripheral units, and input/output interfaces.

Microcontrollers allow for reductions in size, power consumption, and cost of products compared to the microprocessors found in conventional computers. As such, they have enabled the widespread use of computing in a wide range of products and processes. The components typically found in a microcontroller [24].

The figure II.2 details the internal structure of a typical microcontroller, highlighting its processor, memory blocks, and I/O peripherals:

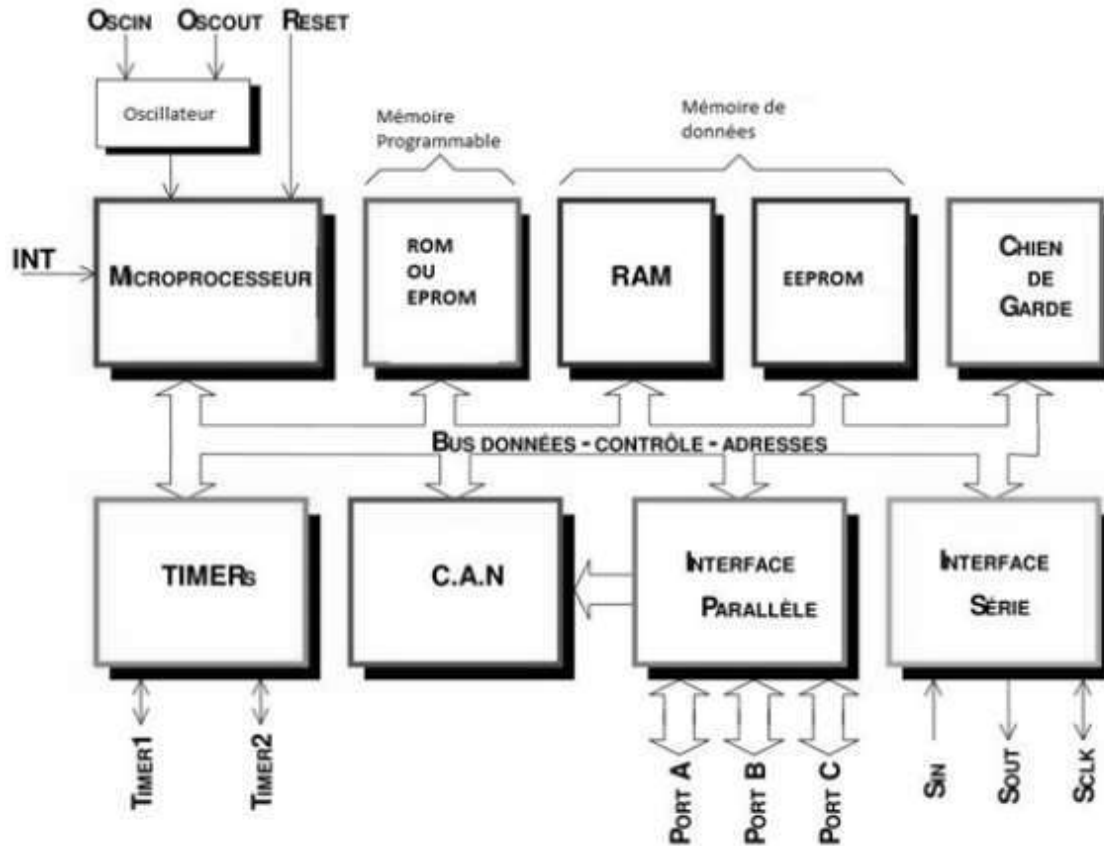


Figure II.2: Internal Components of a Microcontroller [25].

- **Microprocessor:** It is primarily responsible for processing information and data at a rate defined by the internal clock frequency. Its capacity can range from 4 to 64 bits, depending on the selected model.
- **Volatile memory:** Known as RAM, it is a memory with random read and write access. It allows temporary storage of data and variables used during execution.
- **Non-volatile memory:** Known as ROM, it contains the boot program that loads the application from flash memory. Several types of ROM exist: EPROM, EEPROM, etc.
- **Internal oscillator:** It provides the clock signal that regulates the execution rate of the processor's instructions.
- **Input/output interfaces:** These are input-output peripherals that enable the processor to communicate with other external devices.
- **Watchdog:** Ensures that the system does not remain stuck at any particular stage.
- **Other modules:** Including ADC, DAC, Timers, IRQ Controllers, etc.

II.3.2 Advantages

Microcontrollers offer several advantages, including:

- Reduction in hardware size: Achieved through the integration of multiple components.
- Increased system reliability.
- Reduced power consumption.
- Lower manufacturing and design costs [26].

II.3.3 Memories

In an embedded system, various types of memories can be found, which can be classified according to different criteria.

Figure II.3 illustrates the different types of memories, as listed below:

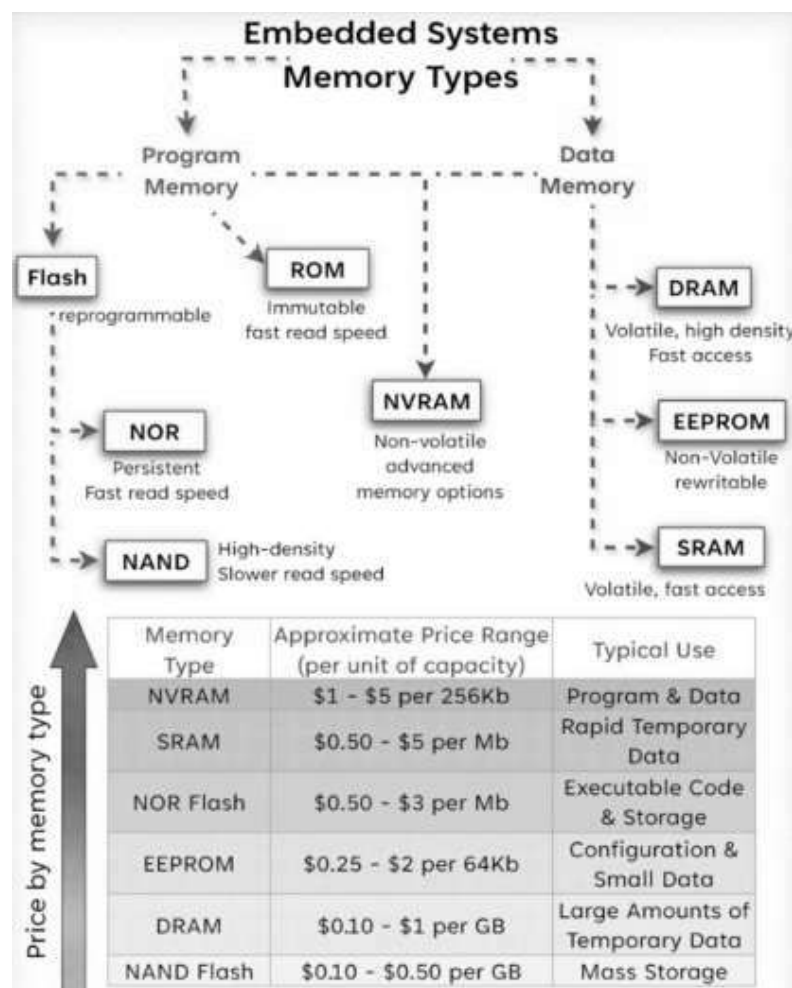


Figure II.3: Types of Memories [27].

Embedded systems use various memory types volatile and non-volatile each suited for different functions like code storage or temporary data. Processor architectures, mainly CISC and RISC, differ in instruction complexity and design, affecting performance and efficiency. The table below summarizes key memory types and highlights main features of CISC and RISC architectures.

Table II.2: Memory Types in Embedded Systems and Processor Architecture Overview.

Category	Type / Aspect	Advantages	Disadvantages	Usage / Notes
Non-Volatile Memory	PROM	Simple, reliable for permanent storage	Cannot be reprogrammed	One-time firmware programming
	EPROM	Reprogrammable via UV light	Degrades after repeated erasures	Early embedded systems
	EEPROM	Electrically erasable & reprogrammable	Slower write speed	Config data, firmware
	FLASH	High storage density, widely used	Wears out with repeated writes	Program storage (firmware, bootloaders)
	NAND Flash	High density, low cost	Slower access time	Mass storage (e.g., SD cards)
	NOR Flash	Fast read access	Lower density	Code execution (e.g., BIOS)
Volatile Memory	SRAM	Very fast, ideal for caches and temporary storage	Expensive, low density	CPU cache, registers
	DRAM	High capacity, cost-effective	Requires refreshing, slower	Main working memory
Processor Architecture	Instruction Complexity (CISC)	Complex instructions requiring multiple clock cycles	More complex decoding, slower execution	Suitable for legacy software, complex tasks
	Instruction Complexity (RISC)	Simple instructions executed in single cycle	Requires more instructions for some tasks	Efficient for high-performance and low power systems
	Number of Registers (CISC)	Few	Limited register availability	Often leads to frequent memory access
	Number of Registers (RISC)	Many	Increased compiler complexity	Reduces memory access, improves speed
	Memory Access (CISC)	Most instructions access memory	Higher memory bandwidth usage	Flexible addressing modes
	Memory Access (RISC)	Only LOAD/STORE instructions access memory	More instructions needed	Simplifies pipeline and execution
	Examples	Intel x86, Pentium, VAX	—	MIPS, ARM, PowerPC, RISC-V

This table outlines the main memory types with their pros, cons, and uses. Non-volatile memories store permanent data, while volatile memories handle fast temporary storage. CISC architectures have complex instructions and fewer registers, whereas RISC architectures emphasize simplicity, speed, and more registers, influencing system design and programming [27] [28].

II.3.4 Choice of a Microcontroller

Microcontrollers come in a wide variety of models, differing in the number of pins, memory size, processing power, and intended application. Some microcontrollers are as compact as having only 6 pins, as shown in Figure II.4, while others are designed for more complex and high-performance systems. Manufacturers offer entire families of microcontrollers, with each family including hundreds of variants tailored for different use cases.

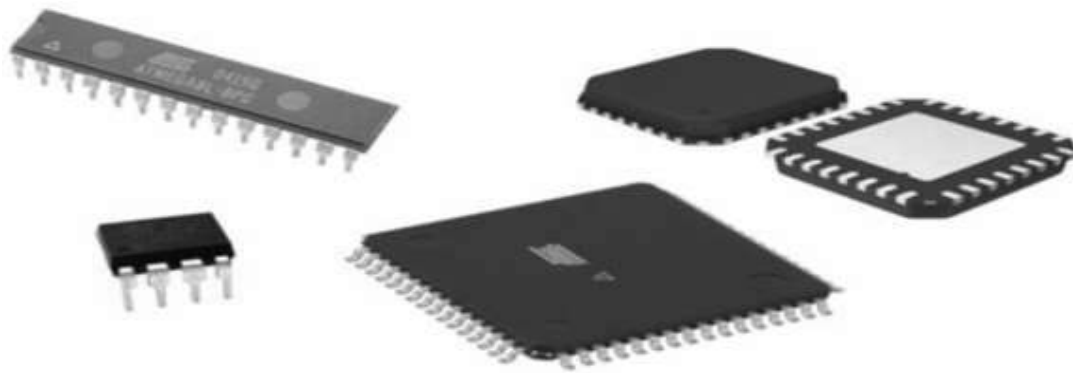


Figure II.4: Examples of Microcontroller Models [29].

II.3.4.1 Key Selection Criteria

Choosing the appropriate microcontroller depends on several technical and practical factors, including:

- **Number of Input/Output Pins:** Determines the capacity to interact with external components and peripherals.
- **Memory Size (RAM & ROM):** Affects the ability to store programs and process data efficiently.
- **Power Consumption:** Includes operating voltage and current requirements, which are critical for battery-powered or energy-sensitive applications.
- **Architecture Compatibility:** The internal structure must align with the specific needs and complexity of the application.
- **Development Environment:** Availability of hardware tools (e.g., programmers, debuggers) and software support (e.g., IDEs, libraries, community resources).

II.3.4.2 Popular Microcontroller Families

Below is a list of some widely used microcontroller families, categorized by their architecture size:

- **8-bit Microcontrollers:**
 - ✓ PIC (Microchip): One of the most popular and widely adopted families.
 - ✓ AVR (ATMEL): Used in platforms like Arduino.
 - ✓ 8051 Derivatives: Including C51, C52 series.
- **16-bit Microcontrollers:**
 - ✓ dsPIC (Microchip): Designed for digital signal processing.
 - ✓ MSP430 (Texas Instruments): Known for ultra-low power consumption.
- **32-bit Microcontrollers:**
 - ✓ AVR32, PIC32, MIPS, PowerPC, ARM, ESP32: Used in advanced applications requiring high processing speed, wireless communication, or real-time control.

II.3.4.3 Selected Microcontroller for This Project

For our implementation, we have selected the ESP32 microcontroller because it satisfies all the previously mentioned criteria:

- Sufficient I/O pins
- Adequate RAM and ROM
- Low power consumption
- Robust internal architecture
- Rich and well-supported development environment (e.g., Arduino IDE, PlatformIO, Espressif tools) [30].

II.4 The ESP32 Microcontroller:

The ESP32 is a low-cost, low-power System on Chip (SoC) microcontroller developed by Espressif Systems. It features integrated Wi-Fi and dual-mode Bluetooth connectivity, making it ideal for modern embedded and IoT applications. First introduced in late 2016, the ESP32 is the successor to the popular ESP8266 microcontroller, offering significantly enhanced capabilities in terms of processing power, memory, and peripheral support.

II.4.1 Resources and Features of the ESP32

The ESP32 is a powerful and versatile platform, widely adopted in embedded system design due to its rich set of integrated resources. Figure II.5 provides a general overview of its main hardware components, including wireless modules and peripheral interfaces.

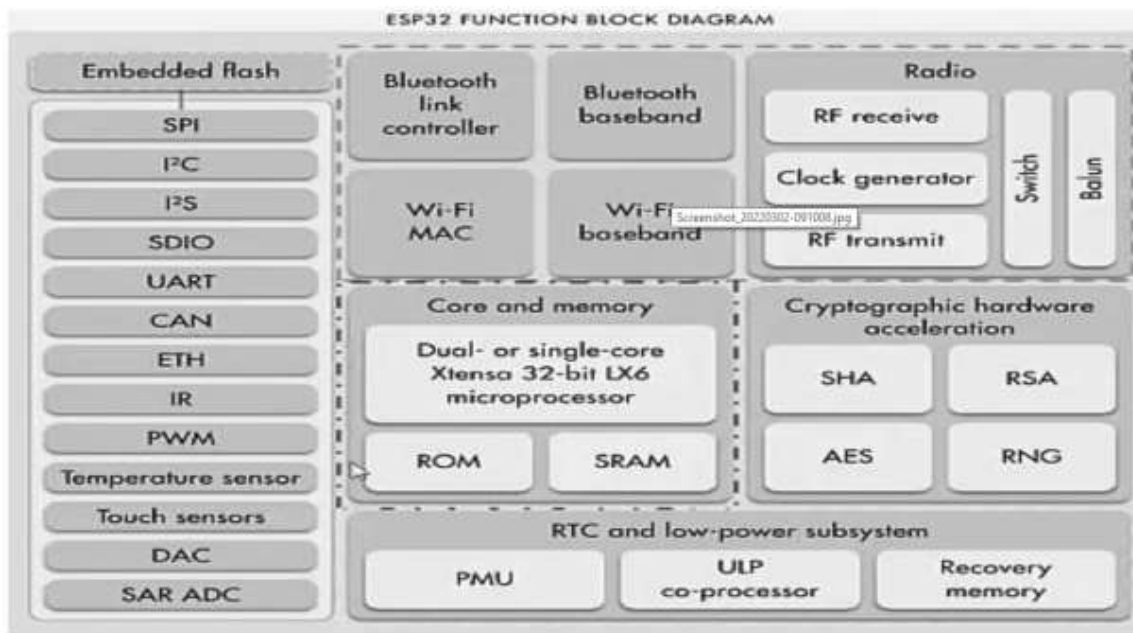


Figure II.5: ESP32 Resources [31].

Table II.3 provides a comparative summary of some widely used microcontroller families. It highlights key features such as architecture type, word size, speed, memory capacity, power consumption, and additional notes on typical usage or limitations. This comparison helps in understanding the strengths and trade-offs of each family, thereby guiding the selection of the most suitable microcontroller for a given application.

Table II.3: Hardware Resources and Functional Features of the ESP32.

Category	Specification / Description
Processors	- Xtensa 32-bit single or dual-core (up to 600 MIPS) - Ultra-low-power coprocessor
Wireless Connectivity	- Wi-Fi - Bluetooth (classic + BLE)
Internal Memory	- 448 KB ROM (boot & core functions) - 520 KB SRAM - 16 KB RTC SRAM (deep sleep)
External Memory	- Up to 16 MB Flash - Up to 8 MB external SRAM
Peripheral Interfaces	- 10 capacitive touch sensors - 18 ADC channels - 2 DAC channels - 2× PC, 2× PS - 3× UART - 1× CAN 2.0 - 4× SPI - 16× PWM - 6× MCPWM (motor control)

II.4.2 Pin Configuration and GPIO Considerations

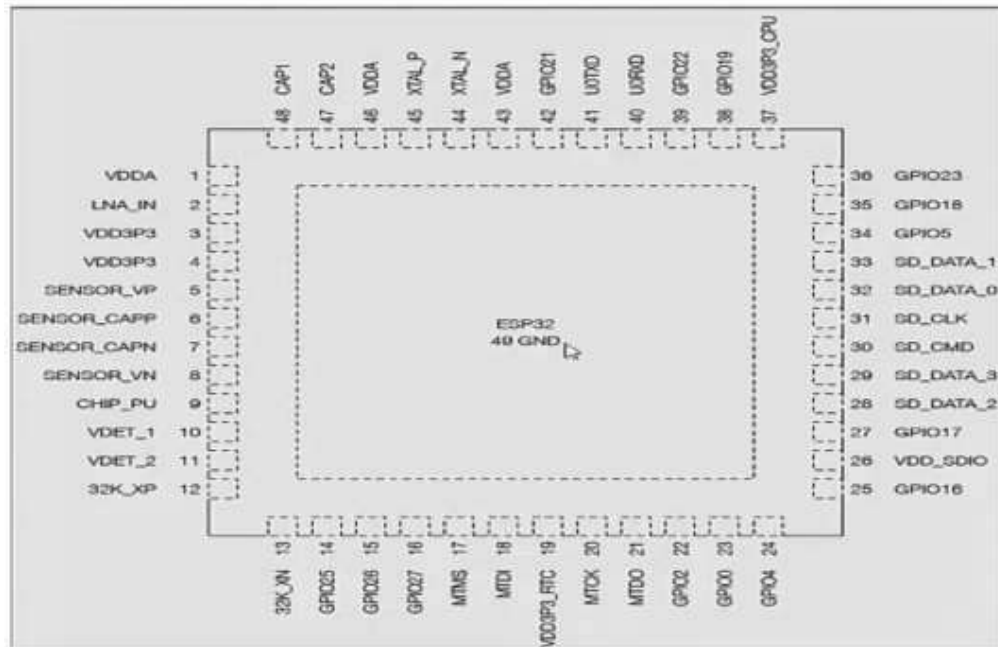
The ESP32 features up to 48 physical pins, many of which are multi-functional, supporting analog, digital, and communication protocols.

Table II.5 outlines the essential characteristics of the ESP32's pin configuration and usage notes.

Table II.4: Summary of ESP32's Built-in Resources and Capabilities.

Feature	Description
Number of Physical Pins	Up to 48 pins
Pin Functions	Most pins are multi-purpose (ADC, DAC, I/O, communication, etc.)
Board-Specific Availability	Not all pins are exposed or usable on all development boards
Usability Notes	Some pins may have restrictions depending on power mode or hardware usage

Figure II.6 illustrates the general architecture of a microcontroller. It shows the main internal components such as the CPU, memory units (RAM, ROM, Flash), input/output ports, timers, and communication interfaces. This simplified diagram helps to visualize how microcontrollers integrate multiple functions into a single compact chip to perform specific embedded tasks efficiently.

**Figure II.6:** ESP32 Microcontroller Pinout [32].

Understanding embedded system memory types, processor architectures, and microcontroller families is essential for designing efficient embedded applications. Each component whether memory or processing unit plays a vital role in determining the system's performance, cost, and power consumption. Selecting the appropriate microcontroller and architecture depends on the specific requirements of the application, including speed, complexity, and resource constraints [31] [32].

II.5 Presentation of the ESP32-CAM Board:

The ESP32-CAM is a development board featuring the ESP32-S chip, an OV2640 camera, several GPIO pins for connecting peripherals, and a micro SD card slot that can be used to store images taken with the camera or for storing files. The ESP32-CAM is an ESP32 development board with its own additional resources:

- OV2640 Camera
- USB-UART Interface
- Reset Button (RST or EN)
- 10 Accessible GPIOs
- 4 MB PSRAM
- Micro-SD Card Interface

This figure displays the ESP32-CAM module, highlighting the main components such as the reset button and microSD slot.

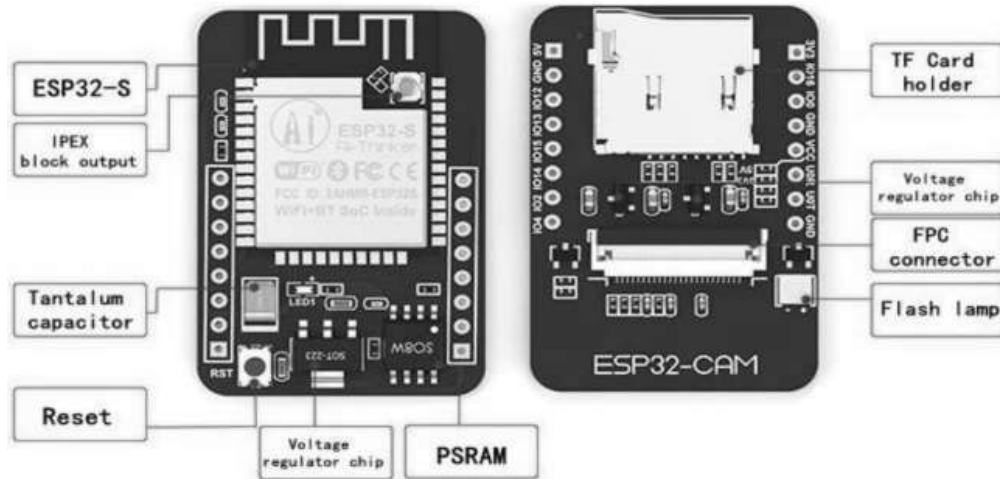


Figure II.7: ESP32-CAM Board [33].

II.5.1 ESP32-CAM Features:

- SPI Flash: Default 32 Mbit
- RAM: 520 KB SRAM + 4 MB PSRAM
- Bluetooth: Bluetooth 4.2 and Bluetooth Low Energy (BLE)
- Wi-Fi: 802.11 b/g/n
- Supported Interfaces: UART, SPI, I2C, PWM
- GPIO Ports: 10
- UART Baud Rate: Default 115200 bps
- Image Output Format: JPEG (only supported by OV2640), BMP, GRAYSCALE
- Frequency Range: 2412 ~ 2484 MHz
- Antenna: Integrated PCB and IPEX connector

- Security: WPA/WPA2/WPA2-Enterprise/WPS
- Power Supply Range: 5V [34].

II.5.2 Pinout of the ESP32-CAM Board:

The figure II.8 shows the specific pin mapping of the ESP32-CAM module, indicating GPIO functions and special features.

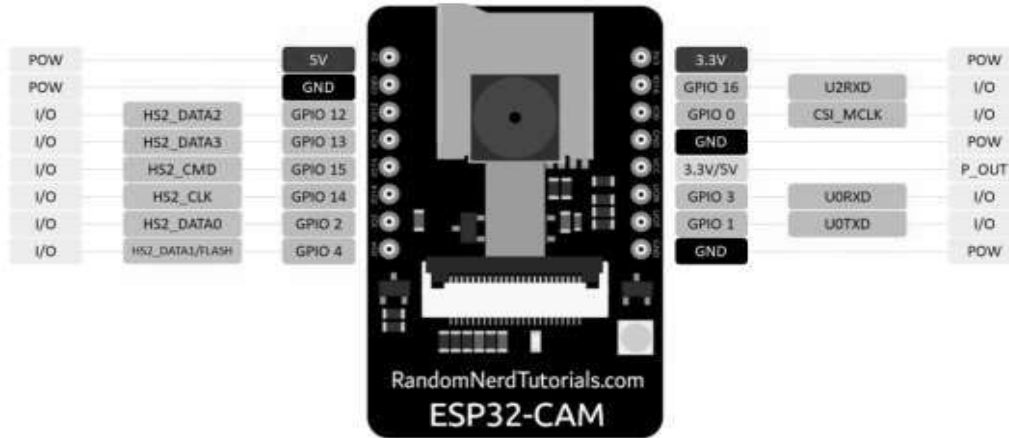


Figure II.8: ESP32-CAM Pinout [35].

There are three GND pins and two power pins: 3.3V and 5V. GPIO 1 and GPIO 3 are the serial pins (TX and RX, respectively). GPIO 0 determines whether the ESP32 is in programming mode. When GPIO 0 is connected to GND, the ESP32 enters programming mode, making it possible to upload code to the board.

II . 6 Study of the ESP32 Library Functions (I/O, Wi-Fi)

II.6.1 Input/Output Functions (GPIO):

One of the most widely used and versatile peripherals for microcontrollers is the GPIO (General Purpose Input/Output), commonly used to read and write the state of I/O pins around the microcontroller. There are two different configuration modes for GPIO:

- Input Mode: In this mode, the GPIO receives the digital state of a specific device, which could be a button or a switch.
- Output Mode: In output mode, the GPIO pin modifies its digital state towards a specific peripheral. It can, for example, control an LED or any other output circuit.

II.6.2 I/O Port Libraires

II.6.2.1 Pin Mode

This function is used to select the operating mode of a specific GPIO pin. `void pin Mode (pin, mode);` // pin defines the GPIO pin number The following modes are supported for basic input and output:

- ✓ INPUT: Defines the GPIO as an input
- ✓ OUTPUT: Defines the GPIO as an output.

- ✓ INPUT_PULLDOWN: Defines the GPIO as an input with an internal pull-down resistor.
- ✓ INPUT_PULLUP: Defines the GPIO as an input with an internal pull-up resistor.

II.6.2.2 Digital Write

The digital Write function sets the state of the selected GPIO pin to either HIGH or LOW. This function is used only if the pin Mode has been set to OUTPUT. `void digital Write (pin, val);` // val defines the digital output state as HIGH or LOW.

II.6.2.3 Digital Read

This function is used to read the state of a specific pin that is configured as INPUT. `int digital Read(pin);`

II.6.2.4 Attach Interrupt

This function is used to attach an interrupt to a specified pin. `Attach Interrupt (pin, void Fun cPtr handler, int mode);` // mode defines the interrupt mode // handler defines the interrupt handler function The following interrupt modes are supported:

- ✓ DISABLED
- ✓ RISING
- ✓ FALLING
- ✓ CHANGE
- ✓ ONLOW
- ✓ ONHIGH
- ✓ ONLOW_WE
- ✓ ONHIGH_WE

II.6.2.5 Attach Interrupt Arg

This function is used to attach an interrupt to a specified pin using arguments. `Attach Interrupt Arg(pin, void Fun cPtrArg handler, arg, mode);` // mode defines the interrupt mode // arg is a pointer to the interrupt arguments [36].

II.6.2.6 Detach Interrupt

This function is used to detach an interrupt from a specific pin. `Detach Interrupt(pin);`

II.6.3 Wi-Fi API

The Wi-Fi API supports the 802.11b/g/n protocol driver. This API includes:

- Station Mode (STA mode or Wi-Fi client mode): ESP32 connects to an access point.
- Access Point Mode (Soft-AP or Access Point mode): Devices connect to the ESP32.
- Security Modes (WPA, WPA2, WEP, etc.).
- Scanning for Access Points

II.6.3.1 Access Point (AP) Mode:

In this mode, the ESP32 is configured as an Access Point (AP), allowing it to receive incoming connections from other devices (stations) by providing a Wi-Fi network. This mode can be used to function as an HTTP or HTTPS server.

This figure II.9 demonstrates how the ESP32 can operate in AP mode, allowing direct Wi-Fi connection from a client device.

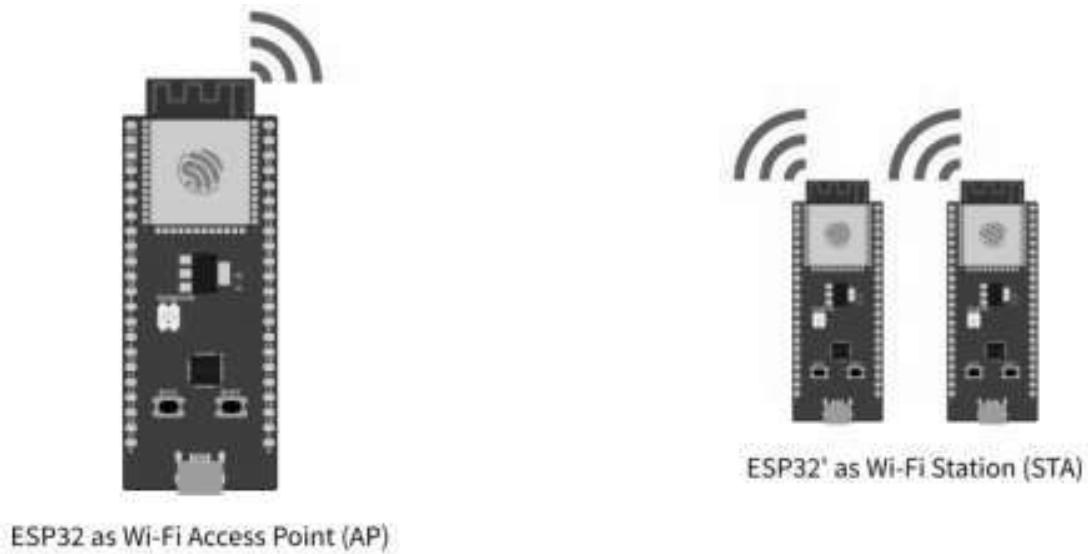


Figure II.9: Access Point Mode Operation [37].

II.6.3.2 Operation in Station (STA) Mode:

The figure II.10 illustrates how the ESP32 connects to an existing Wi-Fi network in station mode to access external servers or services.

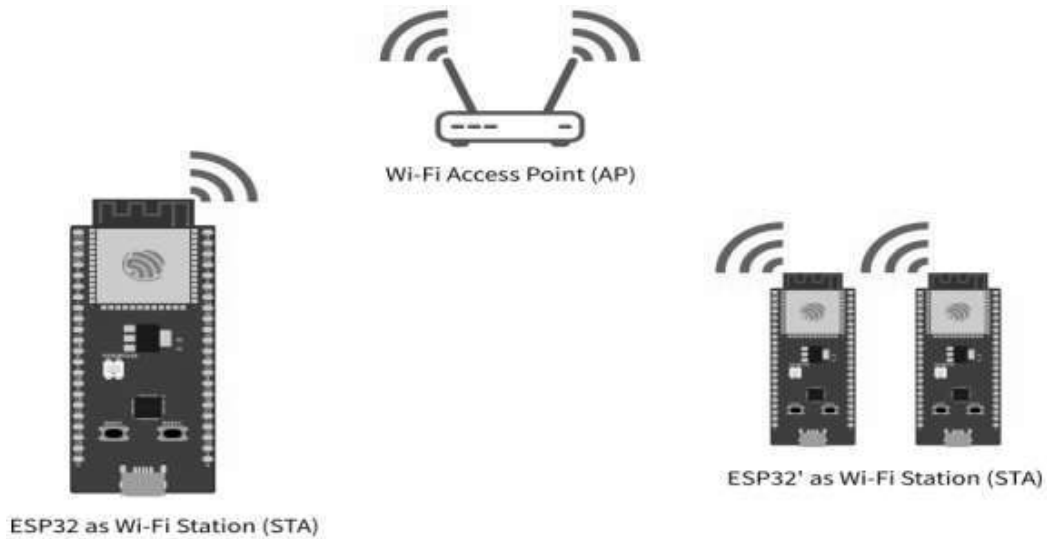


Figure II.10: Operation in Station Mode [37].

The STA mode is used to connect the ESP32 to a Wi-Fi network provided by an access point. This is the mode used to connect the ESP32 board to the internet.

II.6.4 Examples of Wi-Fi Functions:

II.6.4.1 Access Point Functions:

- `WiFi.softAP(ssid, password);` This function allows starting the Wi-Fi in access point mode.
- `bool softAP(const char ssid, const char passphrase = NULL, int channel = 1, int ssid_hidden = 0, int max_connection = 4, bool ftm_responder = false);`
This function allows configuring the characteristics of the access point, where:
- `ssid`: specifies the Wi-Fi SSID. `passphrase`: specifies the Wi-Fi password. If the network is open, set it to `NULL`. `channel`: configures the Wi-Fi channel. `ssid_hidden`: specifies whether the network is hidden. `max_connection`: Configures the maximum number of simultaneous connections. Default is 4. Returns true if the configuration was successful.

II.6.4.2 STA Functions:

- `WiFi.begin(ssid, password);` `ssid` and `password` are the names and password of the network to which the ESP32 is to connect.
- `wl_status_t begin(const char ssid, const char passphrase = NULL, int32_t channel = 0, const uint8_t bssid = NULL, bool connect = true)` [38].

II.6.4.3 Where:

- `ssid`: Specifies the SSID of the network. `passphrase`: Configures the password. Set to `NULL` for an open network. `channel`: Specifies the Wi-Fi channel. `uint8_t* bssid`: Specifies the BSSID of the access point. `connect`: Set to true to automatically connect to the configured network.
- `bool reconnect();` Function that allows the ESP32 to reconnect to the network [32].
- `bool disconnect(bool wifioff = false, bool eraseap = false);` Function that allows the ESP32 to disconnect from the network.

II.6.5 Esp_camera Library:

This is a library used to control the OV2640 camera and is already included in the default ESP32 libraries. To use it, it must be declared in the "include" section at the beginning of the program [34].

```
#include <esp_camera.h>
```

II.6.5.1 Camera_config_t:

This is the instruction provided by the library to configure the camera, and it is "esp_camera_init" which takes a structure of type "camera_config_t" as a parameter. `camera_config_t` Configuration Camera_L;

II.6.5.2 Image Format

```
Config .pixel_ format = PIXFORMAT_JPEG; // Formats: YUV422 / GRAYSCALE / RGB565 / JPEG
```

II.6.5.3 Frame Size:

```
config.frame_size = FRAMESIZE_UXGA; // Frame sizes: VGA / CIF / SVGA / XGA / SXGA / UXGA
```

II.6.5.4 Quality:

Image quality varies from 0 to 63, with a lower number indicating better quality.

```
config.jpeg_quality = 10
```

```
config.fb_count = 2
```

II.6.6. Libraries related to SD card management:

Includes the libraries FS.h and SD_MMC.h for managing the micro SD card.

```
#include "FS.h" // SD Card ESP32
```

```
#include "SD_MMC.h" // SD Card ESP32
```

II.6.6.1 Init Micro SD Card

The function initMicro SD Card initializes the micro SD card and returns an error message if the initialization fails.

```
Serial . println ("starting SD Card");
```

```
if ( !SD_MMC.begin) {
```

```
Serial. println (" SD Card Mount Failed ");
```

```
return ; [39]
```

II.6.6.2 Take Save Photo :

To capture and save a photo, use the function take Save Photo, which accepts the path where you want to save the photo as an argument. void take Save Photo (String path) [40].

II.7. Conclusion

In this chapter, we presented the definition of an embedded system, its characteristics, and its general architecture. We then focused on microcontrollers and, finally, described the microcontroller used in our work, which is the ESP32.

In the next chapter, we will explain in detail the design of the system to be developed and describe all the steps to be followed in the context of our work.

A rectangular area with a marbled, stone-like texture in shades of gray. It has a black border and rounded corners on the left and right sides.

Chapter III: Hardware Implementation of the 3D Vision System

III.1 Introduction

This chapter presents the practical implementation of the 3D camera stabilization and tracking system using the ESP32-CAM. It bridges the gap between theory and practice by transforming the proposed design into a working prototype. The system combines hardware components such as servo motors and the ESP32-CAM module with software algorithms for real-time color detection and camera control.

The ESP32-CAM captures video and communicates with a Python-based GUI that processes the images and detects colored objects. Based on the object's position, the system sends commands to adjust the camera's orientation using a pan-tilt mechanism. This chapter details the hardware setup, wiring, firmware, and software logic, demonstrating the system's ability to track objects dynamically and maintain them centered in the frame.

III.2 Practical Implementation

The practical implementation phase is where the theoretical design of the 3D camera stabilization and tracking system becomes a tangible, functional prototype. This stage involves the careful selection, integration, and testing of hardware components to ensure reliable performance in real-world conditions. The system relies on a synergy between embedded electronics, mechanical structure, and power management to achieve dynamic camera control based on visual input. The following sections present the key components used in the setup, detailing their roles and interconnections within the overall architecture.

III.2.1 Hardware Components

The successful implementation of the 3D camera stabilization and tracking system relies on a set of carefully selected electronic and mechanical components. Each part plays a specific role in enabling real-time image capture, processing, and camera movement control. This section provides an overview of the main hardware elements used in the prototype, highlighting their integration and contribution to the overall system functionality.

The system's core functionality is achieved through the integration of key electronic components. Figure III.1 and Figure III.2 illustrate the two fundamental building blocks of the prototype: the ESP32-CAM module and the MG90S micro servo motor. These components work in tandem to enable image capture, wireless communication, and precise camera movement control.



Figure III.1: ESP32-CAM [41].



Figure III.2: MG90S Micro Servo Motor[42].

The ESP32-CAM is a compact, low-cost development board equipped with a built-in camera and Wi-Fi/Bluetooth connectivity. It serves as the brain of the system, responsible for capturing images, streaming video over a network, and receiving control commands. Programmed using the Arduino IDE, it runs the firmware that handles real-time communication with the host computer and controls the servo motors based on received instructions. Its small size and powerful capabilities make it ideal for embedded vision applications [41].

The MG90S is a high-precision micro servo motor used to control the camera's orientation. It operates within a 0° to 180° angular range and responds to Pulse Width Modulation (PWM) signals generated by the ESP32-CAM. Thanks to its metal gears and compact design, it offers reliable and accurate movement while consuming low power. In this project, two MG90S servos are mounted on a pan-tilt mechanism to enable horizontal (pan) and vertical (tilt) camera adjustments [42].

The system's reliability and development efficiency are improved by two key components: a micro SD card for local data storage and a USB to TTL converter for programming and debugging. Figure III.3 shows the micro SD card used to store captured images, while Figure III.4 depicts the CP2102 USB to TTL converter that facilitates communication between the computer and the ESP32-CAM module during firmware upload and troubleshooting.



Figure III.3: Micro SD Card [44].



Figure III.4: USB to TTL Serial Converter [43].

The micro SD card provides local storage for images captured by the ESP32-CAM, allowing offline analysis and backup of visual data. It is connected to the module via the SPI (Serial Peripheral Interface) protocol and formatted in FAT32 for compatibility. With a capacity of 8GB, it can store thousands of JPEG images depending on resolution and compression. This feature is especially useful in environments with unstable Wi-Fi, ensuring no data is lost during operation [43].

The USB to TTL serial converter acts as a bridge between the computer and the ESP32-CAM, which lacks a built-in USB port. Using the CP2102 chipset, it enables firmware uploading, serial debugging, and power supply during development. The connection is established by linking the converter's TX, RX, and GND pins to their counterparts on the ESP32-CAM, with GPIO0 grounded to enter flashing mode. Once programmed, the converter is disconnected, allowing the system to operate independently [44].

The physical assembly of the system relies heavily on reliable electrical connections and precise mechanical movement. Figure III.5 shows the set of jumper wires used for prototyping the circuit, enabling flexible and reconfigurable connections between components. Figure III.6 illustrates the pan-tilt mechanism, a mechanical structure driven by two SG90 servo motors, which allows the camera to rotate horizontally (pan) and vertically (tilt), providing dynamic control over the field of view.



Figure III.5: Jumper Wires [45].



Figure III.6: Pan-Tilt Mechanism with Dual SG90 Micro Servo Motors [46].

Jumper wires are essential for prototyping electronic circuits without soldering. In this project, male-to-male (M-M) and male-to-female (M-F) cables were used to connect the ESP32-CAM to the servo motors, power supply, and other peripherals on the breadboard. Made of stranded 22 AWG wire with PVC insulation, they offer flexibility, durability, and easy identification through color coding (e.g., red for VCC, black for GND). Their reusability makes them ideal for testing and debugging during development [45].

The pan-tilt mechanism is a lightweight mechanical platform that enables two-axis camera movement. It consists of two SG90 micro servo motors: one controls horizontal rotation (pan), and the other controls vertical tilt. Driven by PWM signals from the ESP32-CAM, the mechanism adjusts the camera's orientation in real time to track moving objects. Its compact size, low power consumption, and high responsiveness make it well-suited for integration into portable or drone-mounted systems, enhancing the system's interactivity and spatial coverage [46].

A stable and continuous power source is crucial for the reliable operation of the camera tracking system. Figure III.7 shows the power module used in the prototype: a standard 5V mobile phone charger. Instead of relying on batteries, this external power supply ensures uninterrupted operation during testing and deployment. The USB cable was cut to expose the positive (red) and negative (black) wires, which were then connected directly to the common power rail on the breadboard.



Figure III.7: power supply [47].

This 5V output supplies power to both the ESP32-CAM module and the two SG90 servo motors. Special care was taken to ensure a shared ground (GND) connection across all components to maintain signal integrity and prevent communication issues. Using a regulated wall charger eliminates voltage drops commonly experienced with AA batteries, providing a clean and consistent power source especially important during high-current operations such as motor activation or image transmission [47].

III.2.2 Electrical Circuit Design

The proper functionality of the 3D camera stabilization and tracking system relies on a well-designed electrical circuit that ensures reliable communication, stable power delivery, and precise control of all components. This section presents the wiring architecture and electrical connections that link the ESP32-CAM module, servo motors, power supply, and programming interface. The circuit is designed to support both operational performance and development flexibility, allowing the system to capture images, process visual data, and dynamically adjust the camera's orientation in real time. The following diagrams illustrate the physical and logical integration of the hardware elements, providing a clear overview of how signals and power are distributed across the system.

III .2.2.1 ESP32-CAM Pan-Tilt System Wiring Diagram

The electrical wiring of the system is a critical component that ensures all hardware elements work together seamlessly to achieve real-time object tracking and camera stabilization. This section provides a detailed overview of the complete electrical connections, highlighting how each component is integrated into the system.

The following Figure III.8 illustrates the comprehensive wiring setup for the ESP32-CAM-based pan-tilt object tracking system. It showcases the interconnections between key components, including the ESP32-CAM module, two micro servo motors (Pan and Tilt), a

power supply, and a USB-to-TTL serial converter used for programming and communication. This wiring configuration enables the system to capture images, process visual data, and dynamically adjust the camera's orientation based on detected objects [48].

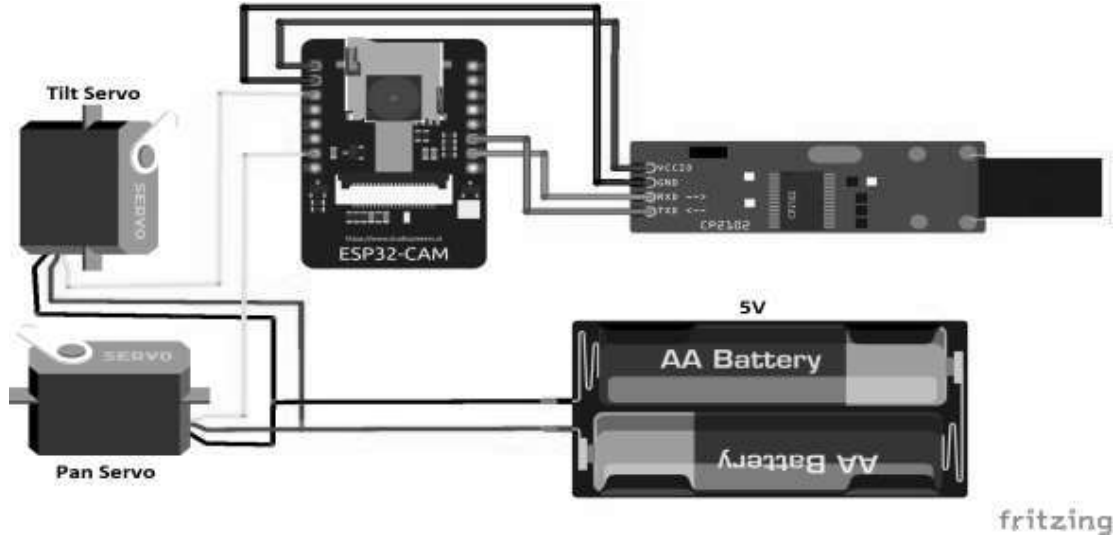


Figure III.8: Wiring Diagram of the ESP32-CAM Pan-Tilt Object Tracking System [48].

A. ESP32-CAM Module

The ESP32-CAM serves as the central processing unit of the system. It is responsible for capturing images, streaming video over Wi-Fi, and controlling the servo motors. The module is connected as follows:

- **Power Supply:** The 5V pin on the ESP32-CAM is connected to the positive terminal of the power supply (5V battery pack).
- **Ground:** The GND pin on the ESP32-CAM is connected to the ground terminal of the power supply.
- **Servo Motors:** Two GPIO pins on the ESP32-CAM are used to control the servo motors:
 - ✓ GPIO Pin for Pan Servo: Connected to the signal (SIG) pin of the Pan Servo motor.
 - ✓ GPIO Pin for Tilt Servo: Connected to the signal (SIG) pin of the Tilt Servo motor.
- **USB-to-TTL Converter:** For programming and debugging purposes, the ESP32-CAM is connected to a USB-to-TTL serial converter via the TX and RX pins, as shown in Figure 9.

B. Micro Servo Motors (Pan and Tilt)

Two micro servo motors are used to enable dynamic camera movement:

- **Pan Servo (Horizontal Movement):** Controlled by PWM signals from one of the GPIO pins on the ESP32-CAM.
- **Tilt Servo (Vertical Movement):** Controlled by PWM signals from another GPIO pin on the ESP32-CAM. Each servo motor has three wires:
 - ✓ Signal (SIG): Connected to a GPIO pin on the ESP32-CAM.
 - ✓ +5V: Connected to the positive terminal of the power supply.
 - ✓ GND: Connected to the ground terminal of the power supply.

C. Power Supply

A 5V power supply (using AA batteries or an external charger) is used to provide stable power to the entire system. The power supply is connected as follows:

- **Positive Terminal (+5V):** Connected to the 5V pins of both the ESP32-CAM and the servo motors.
- **Negative Terminal (GND):** Connected to the GND pins of both the ESP32-CAM and the servo motors. This ensures that all components receive the required voltage and share a common ground for proper operation.

D. USB-to-TTL Serial Converter

A USB-to-TTL serial converter (CP2102-based) is used for programming the ESP32-CAM and enabling serial communication with a computer. The connections are as follows:

- **VCCIO:** Connected to the 5V pin of the ESP32-CAM.
- **GND:** Connected to the GND pin of the ESP32-CAM.
- **TXD (Transmit Data):** Connected to the RXD pin of the ESP32-CAM.
- **RXD (Receive Data):** Connected to the TXD pin of the ESP32-CAM. This setup allows for firmware uploading, serial debugging, and power supply during development [48].

III .2.2.2 USB-to-TTL Serial Communication Interface

The USB-to-TTL serial communication interface plays a crucial role in programming the ESP32-CAM module and enabling real-time debugging and interaction with the system. This section provides a detailed explanation of how the USB-to-TTL converter is connected to the ESP32-CAM, allowing for firmware uploads and serial data exchange between the microcontroller and a computer.

The following Figure III.9 illustrates the wiring connections between the ESP32-CAM and the USB-to-TTL serial converter (CP2102-based). This setup is essential for programming the ESP32-CAM and monitoring its serial output during development and testing phases [49].

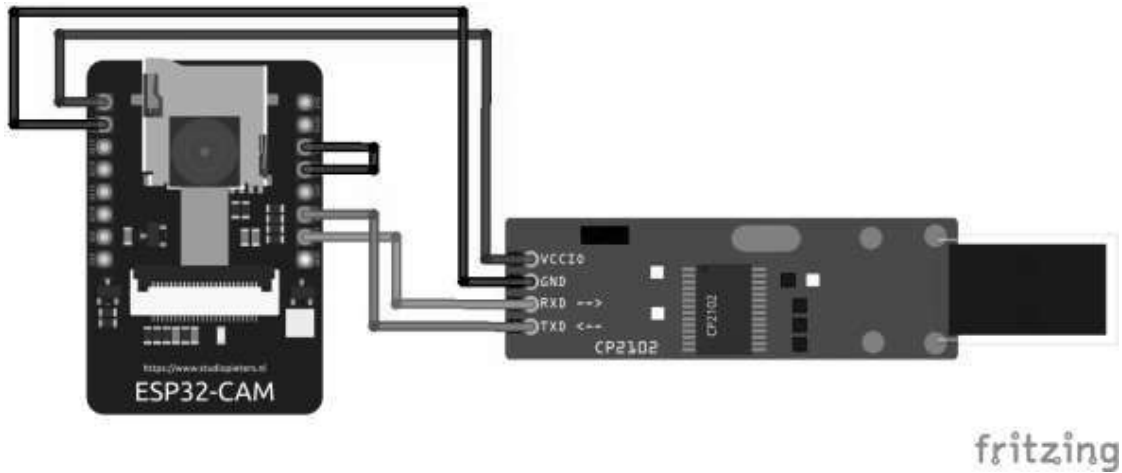


Figure III.9: ESP32-CAM Programming and Serial Communication Interface [49].

A. USB-to-TTL Converter (CP2102)

The USB-to-TTL converter serves as an intermediary device that bridges the USB port of a computer to the UART (Universal Asynchronous Receiver/Transmitter) pins of the ESP32-CAM. It converts USB signals into TTL-level signals that the ESP32-CAM can understand. The key connections are as follows:

- **VCCIO:** Connected to the 5V pin on the ESP32-CAM to provide power.
- **GND:** Connected to the GND pin on the ESP32-CAM to establish a common ground.
- **TXD (Transmit Data):** Connected to the RXD pin on the ESP32-CAM. This allows the computer to send data to the ESP32-CAM.
- **RXD (Receive Data):** Connected to the TXD pin on the ESP32-CAM. This enables the ESP32-CAM to send data back to the computer for monitoring or debugging purposes.

B. ESP32-CAM Module

The ESP32-CAM is equipped with UART pins that allow it to communicate with external devices via serial protocols. In this setup:

- **RXD (Receive Data):** Connected to the TXD pin of the USB-to-TTL converter. This pin receives commands and firmware updates from the computer.
- **TXD (Transmit Data):** Connected to the RXD pin of the USB-to-TTL converter. This pin sends debug information and status updates back to the computer.
- **Power Supply:** The 5V pin on the ESP32-CAM is powered by the USB-to-TTL converter during programming, ensuring stable operation.

C. Ground Connection

A shared ground connection (GND) is established between the USB-to-TTL converter and the ESP32-CAM. This ensures proper signal integrity and prevents any potential issues related to voltage differences.

D. Entering Programming Mode

To upload firmware to the ESP32-CAM, the GPIO0 pin must be grounded temporarily. This puts the module into bootloader mode, allowing it to receive new firmware via the USB-to-TTL converter. After programming, GPIO0 can be left floating or connected to VCC to return to normal operation [49].

III .2.2.3 Servo Motor Connections

The connection of the two micro servo motors (Pan and Tilt) to the ESP32-CAM is a critical aspect of the system, as it enables precise control over the camera's horizontal and vertical movements. This section provides a detailed explanation of how the ESP32-CAM is wired to drive the two servo motors, ensuring smooth and accurate camera positioning.

The following circuit diagram figure III.10 illustrates the electrical connections between the ESP32-CAM and the two micro servo motors. It highlights how PWM signals from specific GPIO pins on the ESP32-CAM are used to control the movement of each servo motor. This setup allows for dynamic adjustment of the camera's orientation based on real-time object tracking data [50].

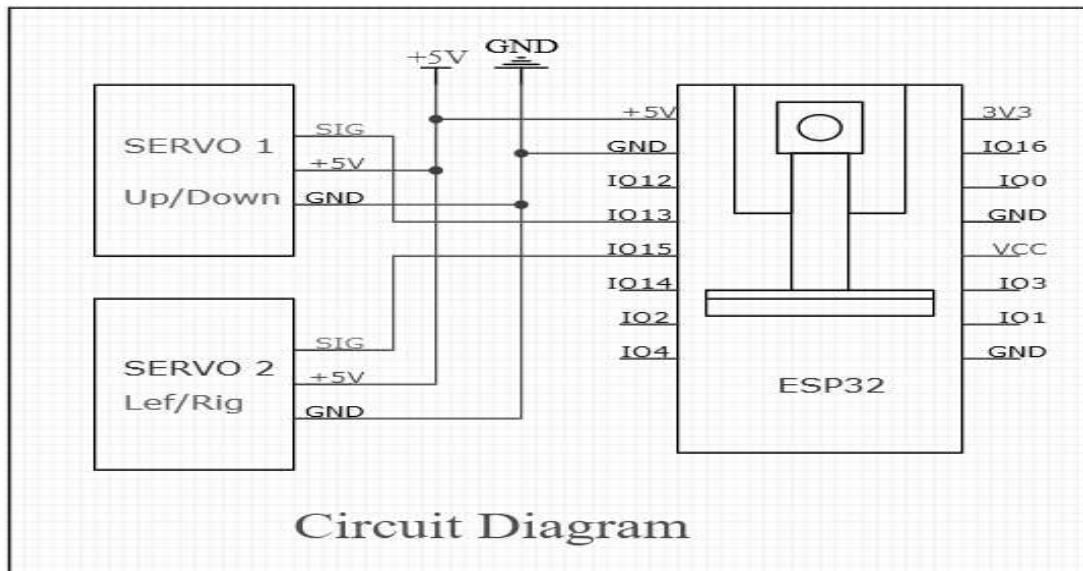


Figure III.10: A diagram showing how to connect the esp32cam in 2 Servo Motor [50].

A. ESP32-CAM Module

The ESP32-CAM acts as the central controller that generates PWM signals to drive the servo motors. Two GPIO pins are dedicated to controlling the two servo motors:

- **GPIO Pin for Pan Servo (Horizontal Movement):** Connected to the SIG pin of the first servo motor (SERVO 1).
- **GPIO Pin for Tilt Servo (Vertical Movement):** Connected to the SIG pin of the second servo motor (SERVO 2).

B. Micro Servo Motors (SERVO 1 and SERVO 2)

Each micro servo motor requires three wires for proper operation:

- **Signal (SIG):** Receives PWM signals from the ESP32-CAM to control the motor's position.
- **+5V:** Provides power to the motor.
- **GND:** Establishes a common ground connection.

SERVO 1 (Up/Down - Tilt Servo):

- **SIG:** Connected to one of the GPIO pins on the ESP32-CAM (e.g., GPIO 12 or another designated pin).
- **+5V:** Connected to the +5V pin on the ESP32-CAM.
- **GND:** Connected to the GND pin on the ESP32-CAM.

SERVO 2 (Left/Right - Pan Servo):

- **SIG:** Connected to another GPIO pin on the ESP32-CAM (e.g., GPIO 13 or another designated pin).
- **+5V:** Connected to the +5V pin on the ESP32-CAM.
- **GND:** Connected to the GND pin on the ESP32-CAM.

C. Power Supply

Both servo motors share the same power supply:

- **+5V:** Provided by the ESP32-CAM's +5V pin, which is connected to an external power source (such as a battery pack or regulated power supply).
- **GND:** Connected to the GND pin on the ESP32-CAM to ensure a common ground reference.

D. Ground Connection

A shared ground (GND) is established between the ESP32-CAM and both servo motors. This ensures consistent voltage levels and prevents any potential issues related to grounding.

E. PWM Signal Generation

The ESP32-CAM generates PWM signals on the designated GPIO pins to control the position of each servo motor. The duty cycle of the PWM signal determines the angle at which the servo motor will rotate:

- **Duty Cycle Range:** Typically, between 0% and 100%, corresponding to the full range of motion (0° to 180°) for most micro servos.
- **Control Logic:** The firmware running on the ESP32-CAM calculates the required PWM values based on the detected object's position and sends them to the appropriate GPIO pins [50].

III.3 Software Part

The software component of the 3D camera stabilization and tracking system is responsible for enabling intelligent behavior through real-time image processing, decision-making, and actuator control. This section presents the architecture and functionality of the software system, which is divided into two main parts: the embedded firmware running on the ESP32-CAM microcontroller, and the host-side application developed in Python for object detection and command generation.

The system operates in a client-server model: the ESP32-CAM acts as a server, hosting a web interface, capturing images, and controlling the servo motors. Meanwhile, the Python application runs on a host computer as a client, receiving video frames, analyzing them using OpenCV, and sending directional commands to adjust the camera's position based on the detected object's location.

Key aspects such as code structure, communication protocol (HTTP), and control logic are explained in detail. Additionally, the flowcharts and core functions of both the Arduino (ESP32) and Python codes are described to illustrate how the system achieves dynamic object tracking and camera stabilization. This integration of hardware and software demonstrates a fully functional, low-cost smart vision system capable of real-time response.

III.3.1 Integrated Object Tracking System Using PC Client and ESP32-CAM

This system is composed of two interconnected programs working in real time to achieve automated object tracking. The PC client acts as the control center, retrieving live images from the ESP32-CAM, processing them to locate the target, and sending movement commands based on calculated position errors. The ESP32-CAM, functioning as a server, receives these commands, adjusts its servos to reposition the camera, and continuously provides updated images back to the client. Together, they form a closed-loop feedback system where the PC provides decision-making intelligence and the ESP32-CAM delivers physical motion control, ensuring the tracked object remains centered in the camera's view.

III.3.1.1 PC Client Side – Object Tracking and Command Sending

This flowchart illustrates in figure III.11 the client-side program running on a personal computer, which is responsible for capturing live images from the ESP32-CAM, processing them using computer vision techniques, and determining the position of a target object. Based on this analysis, the program decides the movement direction and sends corresponding control commands to the ESP32-CAM over an HTTP connection [51].

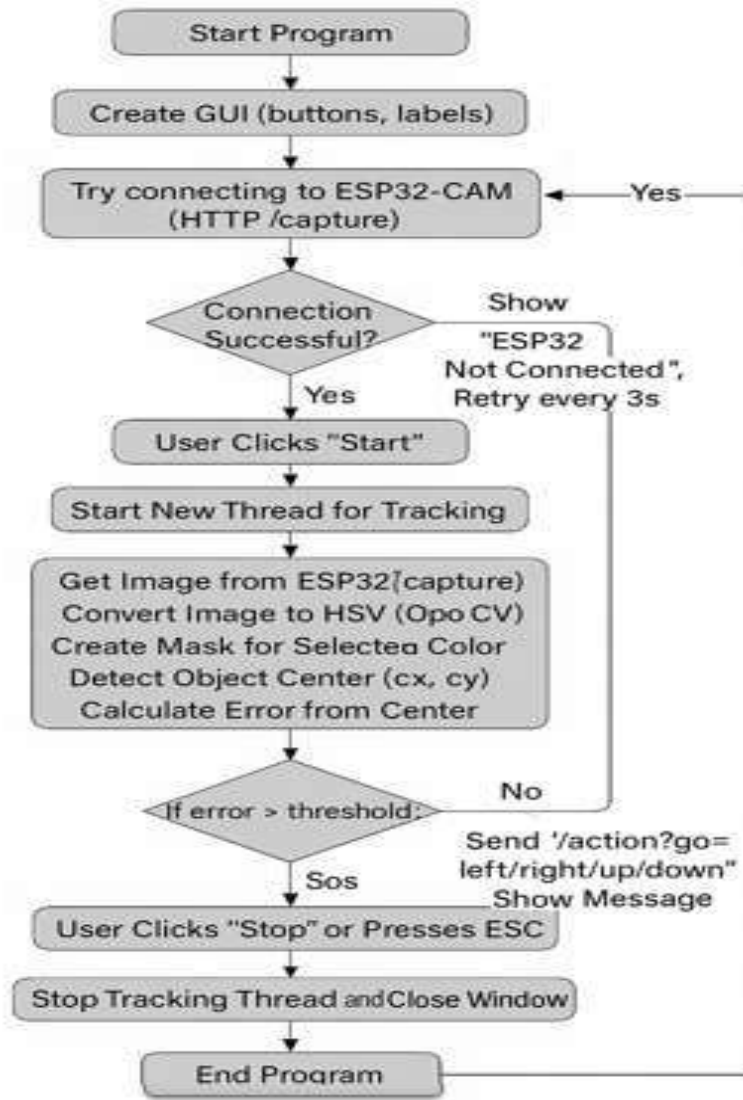


Figure III.11: ESP32-CAM (Flowchart: Object Tracking Program using ESP32-CAM) [51].

A. Flowchart explanation

This flowchart describes a desktop application whose job is to capture images from an ESP32-CAM, process them with OpenCV to detect and track an object by color, and send motion commands back to the ESP32. The application starts by building a simple GUI (buttons, labels, status indicators). The GUI gives the user control (Start / Stop) and visual feedback (connection status, possibly the live frame and current HSV ranges). The GUI is also the main thread that keeps the window responsive while other work runs in the background.

After GUI creation the program repeatedly attempts to connect to the ESP32-CAM using an HTTP capture endpoint (for example `http://<esp_ip>/capture` or `/cam-lo.jpg`). If the connection fails, the GUI shows a clear “ESP32 Not Connected” message and the client retries

periodically (the diagram shows every 3 seconds). This reconnection loop protects the user experience so the app doesn't crash when the device or network is temporarily unavailable.

This System Interface During Connection Attempt shows in Figure 12 the initial state of the color tracking system, where the software is attempting to establish a connection with the ESP32-CAM module. The message "Connecting to ESP32..." indicates that the system is waiting to receive a response from the device. The "Open to start tracking" button is disabled until the connection is successful, ensuring that tracking only begins when communication is stable.



Figure III.12: System Interface During Connection Attempt.

Once the connection is established, the interface evolves into a more interactive form, as shown in figure 13. The user can now select the target color to track (Blue, Red, Green, or Yellow). In this example, the Blue button is highlighted, indicating it has been selected. This enhanced interface allows for dynamic control over the tracking process, enabling the system to focus on a specific color after connection is confirmed.



Figure III.13: User Interface with Color Selection After Successful Connection.

Just pick a color spawns a new tracking thread (a background worker) so the image capture and processing loop won't block GUI events. Separating GUI and image processing in different threads avoids freezes and keeps the interface responsive to Stop, ESC, or slider adjustments.

Inside the tracking thread there is a continuous processing loop: the client gets an image frame from the ESP32 over HTTP, converts the frame from BGR/RGB to HSV color space (HSV is used because hue separates color independently of lighting), and builds a binary mask using user-selected HSV bounds (via GUI trackbars). Typical follow-ups include morphological filtering (open/close) to remove noise, then contour detection to find candidate objects.

The algorithm usually picks the largest contour (or the most circular) and computes its centroid (cx, cy) or bounding box center. Once the object center is known, the program calculates the error between the object's center and the image center (e.g., $\text{error_x} = \text{cx} - \text{frame_center_x}$, $\text{error_y} = \text{cy} - \text{frame_center_y}$). The flowchart uses a threshold test: if the magnitude of this error exceeds a configured threshold, the client decides corrective action is required. That action is mapped to simple directional commands (left/right/up/down) depending on the sign and magnitude of error_x and error_y . When the threshold is exceeded, the client sends an HTTP command such as `/action?go=left` to the ESP32, and optionally shows a message or indicator (e.g., “moving left”). If the error is within the threshold, no correction command is sent and tracking continues.

The loop continues until the user clicks Stop or presses ESC. When stop is requested, the GUI signals the tracking thread to terminate (using a thread-safe flag), the thread completes its current cycle, all resources (network sockets, windows) are cleaned up, and the program exits. Proper thread shutdown and `join()` behavior avoid dangling threads or crashes.

B. Practical implementation notes and robustness considerations

These improvements make the system stable, responsive, and safe in real deployments.

- Use short HTTP timeouts and catch network exceptions so the thread can recover or display a connection error.
- Limit command send rate (debounce) to avoid flooding the ESP32.
- Consider smoothing the error signal (moving average) or using a simple PID controller to convert error to servo steps for smoother motion.
- Keep image resolution modest for real-time speed.
- Secure the endpoints if used on an untrusted network (basic token or local network only).

The client-side system acts as the “brain” of the operation, performing advanced image analysis and making real-time movement decisions. It ensures the ESP32-CAM receives precise movement instructions to keep the tracked object centered in the camera’s view [51].

III.3.1.2 ESP32-CAM Server Side – Command Reception and Execution

This flowchart in figure 13 represents the firmware running directly on the ESP32-CAM. Its role is to act as a Wi-Fi-connected HTTP server that receives commands from an external client (such as the PC application in the first flowchart), and either returns responses or controls servo motors accordingly.

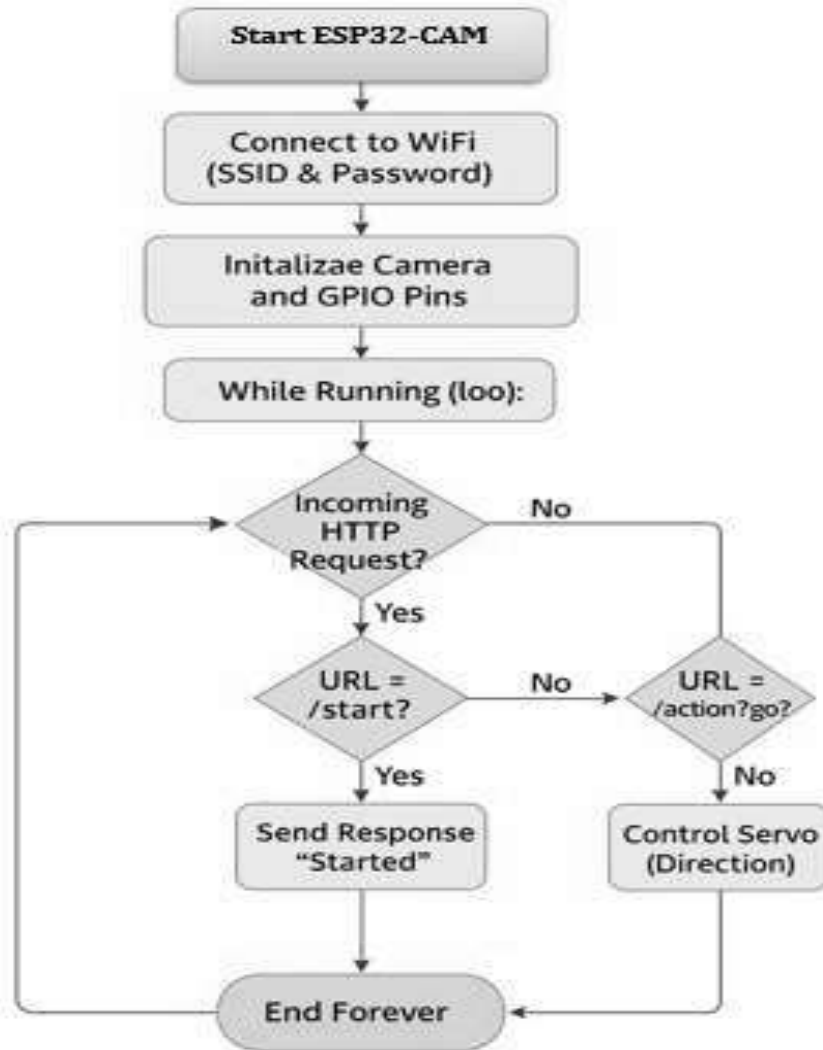


Figure III.14: ESP32-CAM Server Side – Command Reception and Camera Control [52].

The program begins with the Start ESP32-CAM step, which marks the device's initialization process after power-up or reset. The first operational task is to connect to a Wi-Fi network using predefined SSID (network name) and password stored in the code. This is essential because all communication with the controlling PC is done over TCP/IP via HTTP requests, so the ESP32-CAM must be on the same network as the client.

Once the network connection is established, the firmware initializes the camera module (configuring resolution, frame format, frame buffers) and sets up GPIO pins for any connected

hardware such as servo motors used for pan/tilt movement. At this stage, the device is ready to capture images and to move mechanical parts on demand.

The main logic runs in a continuous loop (`loop()` function in Arduino terminology). In every iteration, the ESP32-CAM checks if there is an incoming HTTP request from any client. This is handled by the built-in web server library (such as `WebServer` or `ESPAsyncWebServer`), which listens for requests on specific URLs (endpoints). If no request is present, the device simply continues looping and waits.

If an HTTP request is detected, the firmware checks the URL path to determine the intended action. The first check is whether the URL equals `/start`. This endpoint is used by the PC client to signal the start of tracking operations. When such a request arrives, the ESP32-CAM sends a simple HTTP response back to the client containing a message like "Started". This confirms that the server is active and ready to receive motion commands.

If the request is not `/start`, the firmware checks whether it matches `/action?go=`. This endpoint is designed to carry movement instructions. The query parameter `go` specifies the desired direction left, right, up, or down. When such a request arrives, the ESP32-CAM executes servo control routines to physically move the camera mount in the specified direction. The GPIO pins connected to the servo motors are pulsed with the correct PWM signals to achieve the intended movement.

After processing any valid request, the loop continues, always listening for the next incoming HTTP command. This design allows the ESP32-CAM to remain responsive indefinitely, enabling real-time remote control by the client application.

From an implementation standpoint, this server logic is minimal yet efficient. It delegates all image processing and decision-making to the PC side, keeping the firmware lightweight and focused on hardware control and basic HTTP request handling. This separation of concerns also simplifies debugging: network issues, image analysis, and tracking logic are handled on the PC, while the ESP32 deals only with movement and connectivity.

The server-side program acts as the “muscle” of the system, executing movement commands from the client. By combining camera functionality with servo control, it enables precise physical adjustments that align with the object tracking results determined by the PC [52].

III.3.2 Explanation of Key Functions in the System

To ensure the proper operation of the 3D camera stabilization and tracking system, several critical functions are implemented in both the Arduino firmware (running on the ESP32-CAM) and the Python-based host application. These functions handle essential tasks such as image capture, communication, servo control, and object detection. This section provides a

concise explanation of the most important functions used in the code, highlighting their role, input/output behavior, and contribution to the overall system functionality. Understanding these functions is key to comprehending how the hardware and software components interact to achieve real-time color tracking and camera movement.

III.3.2.1 First Flowchart – PC Client Side

- **create_gui()**

Creates the graphical interface of the program, including buttons, labels, and status indicators. It also sets up any interactive controls for the user, such as start/stop buttons and color adjustment sliders.

- **connect_to_esp32()**

Attempts to establish a connection to the ESP32-CAM over HTTP. It sends a request to the camera's endpoint to verify whether it is reachable.

- **capture_image()**

Retrieves the latest image frame from the ESP32-CAM through the network. This frame is prepared for further image processing.

- **process_image_hsv()**

Converts the captured image from its default color format into HSV color space. This format is chosen because it separates color information from brightness, making it more reliable for color detection.

- **create_color_mask()**

Generates a binary mask that highlights only the pixels within the specified color range. This allows the program to isolate the object of interest.

- **detect_object_center()**

Analyzes the mask to locate the object's position in the frame. It determines the coordinates of the object's center.

- **calculate_error()**

Measures the difference between the object's center position and the center of the frame. This value is used to decide if corrective movement is needed.

- **send_movement_command()**

Sends a command to the ESP32-CAM, instructing it to move the camera in a specific direction based on the calculated error.

III.3.2.2 Second Flowchart – ESP32-CAM Server Side

- **setup()**

Runs once when the ESP32-CAM starts. It initializes the Wi-Fi connection, the camera module, and the servo motor control pins.

- **WiFi.begin()**

Connects the ESP32-CAM to the designated wireless network, enabling communication with the PC client.

- **initCamera()**

Configures the camera settings such as resolution, frame format, and other operational parameters.

- **initServo()**

Prepares the servo motors for movement by attaching them to specific control pins and setting their initial positions.

- **server.on("/start")**

Defines the action to take when the /start request is received. Typically, it sends back a confirmation message to the client indicating that the server is active.

- **server.on("/action")**

Defines how the ESP32-CAM processes incoming movement commands. It reads the direction parameter and calls the appropriate movement routine.

- **loop()**

Continuously checks for incoming HTTP requests and handles them in real time, ensuring the device responds immediately to client commands.

- **moveServo(direction)**

Moves the servo motor to adjust the camera position according to the direction received from the client.

The complete system described combines the processing power of a PC client with the embedded control capabilities of the ESP32-CAM to achieve efficient, real-time object tracking. The PC application handles image acquisition, color-based object detection, and decision-making, while the ESP32-CAM executes the corresponding movement commands to adjust the camera's position. The two flowcharts illustrate this interaction from both sides—client and server—showing a continuous feedback loop between image analysis and physical camera control. By understanding the structure, functions, and logical order of these components, the relationship between software intelligence and hardware execution becomes clear, demonstrating how both parts work in harmony to keep a moving object centered in view [51] [52].

III.4 System Results and Dynamic Operation

Since the 3D camera stabilization and tracking system is inherently dynamic—constantly adjusting the camera’s position in response to moving objects—the results cannot be represented by static measurements alone. Instead, the system’s performance is best demonstrated through a sequence of images captured during real-time operation. These images illustrate the different stages of object detection and camera adjustment, showing how the system detects a colored object, calculates its position relative to the frame center, and commands the pan-tilt mechanism to re-center the target.

The following section presents a series of operational snapshots that capture the system in action. Each image reflects a specific phase in the tracking process, providing visual evidence of the successful integration between hardware and software components. This step-by-step visualization confirms the system’s responsiveness, accuracy, and ability to maintain the tracked object at the center of the camera’s field of view.

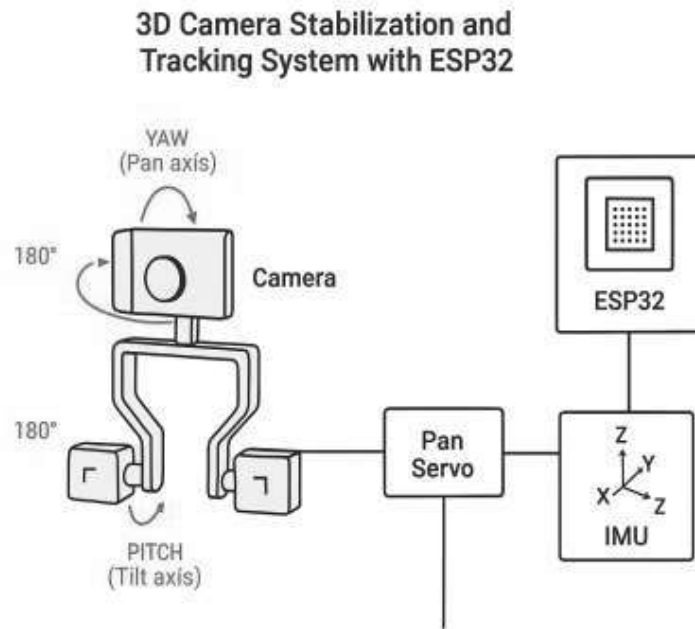


Figure III.15: 3D Camera Stabilization and Tracking System with ESP32 (Pan-Tilt 180° Support)

III .4.1 A final image of the project

This figure III.14 present the complete physical assembly of the 3D camera stabilization and tracking system

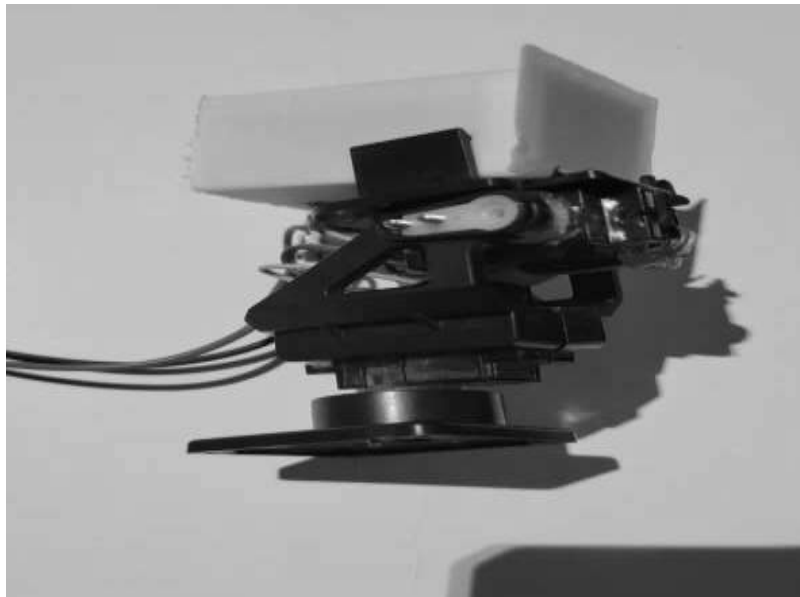


Figure III.16: Final form of the project.

This Figure showcases the physical implementation of a visual tracking system project, featuring an ESP32-CAM camera module mounted on a versatile Pan-Tilt mechanism. The Pan-Tilt structure is driven by two servo motors: one enabling horizontal (pan) movement and the other facilitating vertical (tilt) motion. This mechanical configuration, combined with intelligent programming, allows the camera to automatically adjust its orientation to follow target objects or colors within its field of view. This setup is fundamental for applications requiring dynamic observation or visual tracking, such as smart surveillance systems, simple robotic vision, or advanced home automation projects.

III .4.2 Electrical components of a circuit

The Figure 15 illustrating the electrical component connections serves. This Figure provides of how various electronic components such as the ESP32-CAM board, servo motors, power source, and the CP2102 programming module are interconnected to form a functional system.



Figure III.17: Circuit installation.

The following table .1 provides a detailed overview of the main hardware components used in the ESP32-CAM based object tracking system. Each element is described in terms of its physical appearance and specific role within the setup. This structured breakdown clarifies how the components ranging from the camera module to the servo motors and the laptop interface work together to achieve dynamic object detection and tracking.

Table .1: System Components and Their Roles in the ESP32-CAM Based Object Tracking Setup.

Component	Appearance	Function
Camera Module (ESP32-CAM)	White 3D-printed enclosure at the top with a visible small camera lens at the front.	Captures images and videos for image processing, object detection, and tracking. It acts as the "eye" of the system.
Servo Motors	Two small motors (blue and black) located below the camera. Wires (red, black, yellow/orange) are connected.	Control the movement of the camera: • Pan Servo for horizontal rotation. • Tilt Servo for vertical tilt.
Mounting Bracket / Frame	Black plastic parts connecting the camera to the servos and to the wooden base.	Provides mechanical structure and stability, ensuring proper alignment and coordinated motion between camera and servos.
Electrical Wires	Colored wires (red, black, yellow) extending between servos and ESP32-CAM.	• Red: Power (VCC). • Black: Ground (GND). • Yellow/Orange: Signal/control. Used for powering and controlling servos from the ESP32-CAM.
Wooden Base	Rectangular wooden platform at the bottom of the setup.	Acts as a stable foundation to prevent vibrations or falling during servo movements.
Laptop Computer (ASUS)	Laptop visible near the device.	• Programming: Uploading code to ESP32-CAM. • Monitoring: Viewing live feed and tracking data through GUI. • Power Source: Possibly powers the device via USB.
USB Cable	White cable running from laptop to the base of the device.	• Provides power to ESP32-CAM. • Enables serial communication (via USB-to-TTL like CP2102) for programming and real-time data monitoring.

III.4.3 System Operation and Experimental Validation

In this section, we move from theoretical concepts to real-world implementation. Based on the experimental design and working principles described earlier, we will demonstrate how the 3D camera stabilization and tracking system actually performs. The ESP32-CAM will capture live video, which is then processed by a Python-based GUI using OpenCV to detect a

colored object. Once detected, the system calculates the object's position relative to the center of the frame and sends directional commands (left, right, up, down) to adjust the camera's orientation via servo motors.

This closed-loop process—capture → process → decide → act—will be visually demonstrated through a sequence of images showing the camera dynamically tracking the object until it is centered.

We will verify the ESP32-CAM-based tracking system through a controlled experiment using a blue-colored tube as the target object. Four images will be captured to demonstrate the system's performance before and after object detection.

III .4.3.1 Before Detection:

This figure III.20 shows the system scanning its surroundings before detecting any object. The left image displays the ESP32-CAM setup, while the right image shows the live camera feed without any target detected.

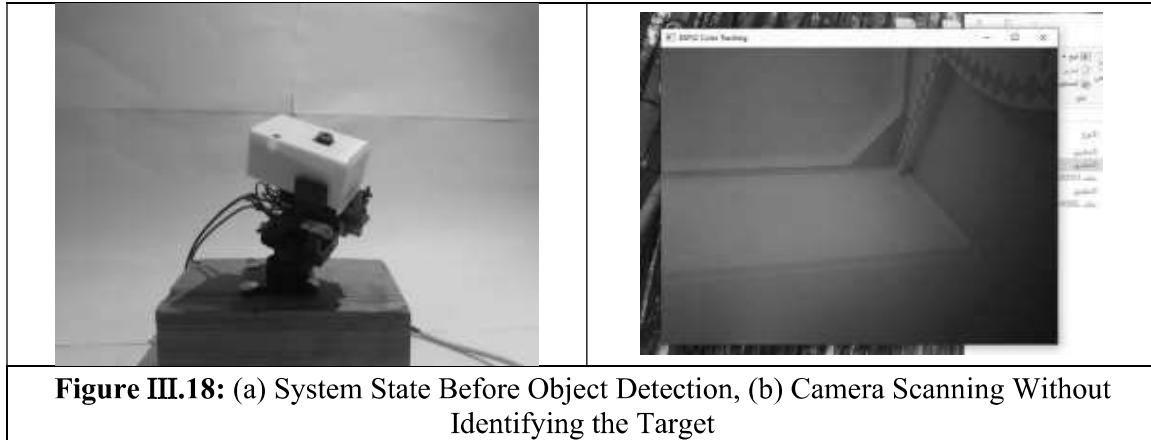


Figure (a) shows the physical state of the system as it actively scans its surroundings. The camera module is mounted on a pan-tilt mechanism, with the servo motors in motion as the device searches for the target.

Figure (a), taken from the computer interface, displays the camera's real-time feed, which shows the environment without detecting any dominant colored object yet. This phase confirms that the system is in continuous tracking mode awaiting a target.

III .4.3.2 After Detection:

This figure III.21 shows the system after detecting a blue tube. The left image displays the device turned toward the object, and the right image shows the camera feed with a green dot marking the detected blue color.

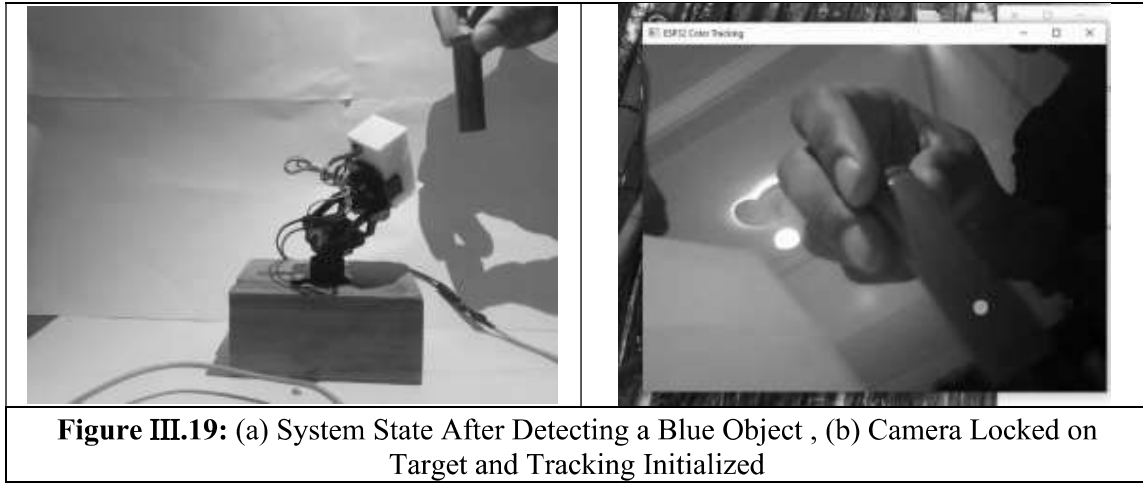


Figure (a) presents the system after the introduction of the blue tube into the field of view.

The camera and servos have reoriented to lock onto the detected object.

Figure (b) reveals the camera feed from the ESP32-CAM module, where the system has successfully identified the blue object and marked it with a green dot, indicating the center of the detected target. This overlay confirms the proper operation of the object detection and tracking algorithm.

.4.4 Steps of Target Detection and Tracking Process

The primary objective of this system is to autonomously detect and track a predefined target using embedded vision and servo control. To achieve this, the device employs a 3D camera stabilization and tracking mechanism that integrates real-time video capture with intelligent image processing algorithms. The process begins with the camera actively scanning its surroundings in search of the target. Once the target is detected, the system dynamically adjusts the camera's orientation via servo motors to maintain the object at the center of the frame. The figure III.18 below provides a visual overview of the key stages in this process:

- The initial setup where the camera is unaligned with the target.
- The active scanning phase, the device moves to search for the target.
- The final stage, where the camera successfully detects the target, and the device marks the desired color to be tracked with a green dot, demonstrating the effectiveness of the tracking algorithm.

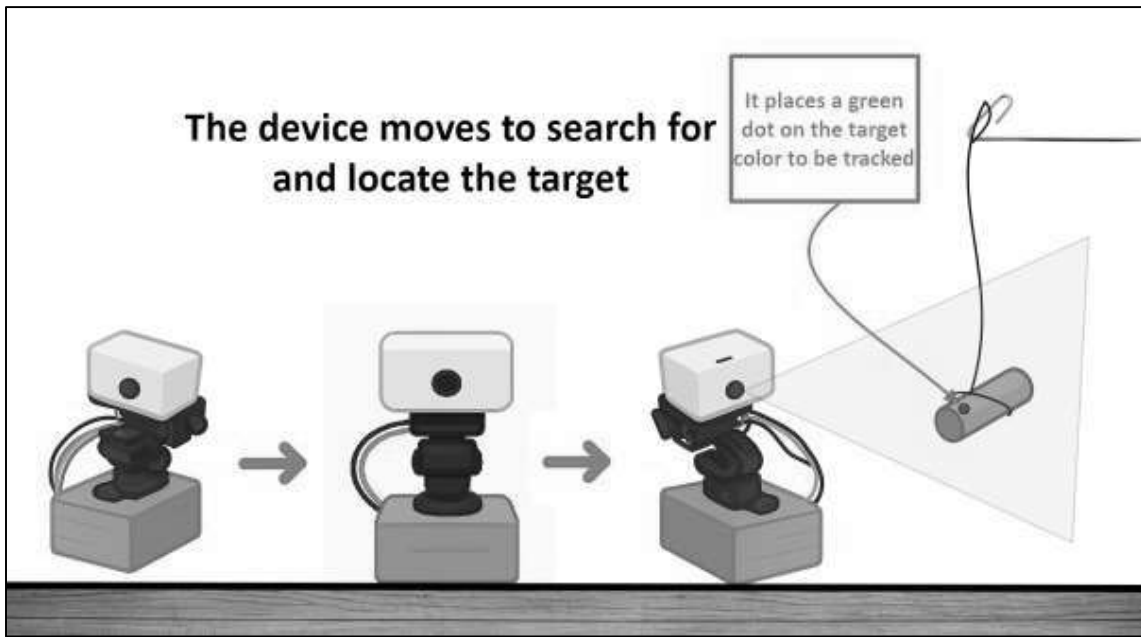


Figure III.20: Target Detection and Tracking Process.

This sequence highlights the seamless integration of hardware and software components, showcasing how the system transitions from passive scanning to active, real-time tracking. The following sections will delve deeper into the experimental setup, implementation details, and validation results.

III .4.4.1 Step 1: Initial Search Mode (Scanning Phase)

The system begins in a scanning state, where the camera is not yet aligned with any target. The ESP32-CAM, mounted on a pan-tilt platform driven by two servo motors (one for horizontal pan, one for vertical tilt), starts sweeping the environment.

The camera moves systematically—typically from side to side (panning)—to cover a wide field of view and increase the likelihood of detecting the target object.

During this phase, no object has been identified, and the system remains in a continuous search loop.

III .4.4.2 Step 2: Object Detection

As the camera pans, it captures live video frames and streams them over Wi-Fi to a Python-based client application running on a host computer. The client uses OpenCV to process each frame in real time, converting the image from BGR to HSV color space for more accurate and lighting-invariant color detection.

A predefined HSV range for the blue color is used to create a binary mask, isolating regions that match the target. Morphological operations (such as erosion and dilation) are applied to reduce noise, followed by contour detection to identify the largest blue object. Once a valid object is found, its centroid (center point) is calculated.

III .4.4.3 Step 3: Target Acquisition and Camera Adjustment

The system compares the object's centroid with the center of the image frame. If the object is off-center, a positional error is calculated in both the horizontal (x-axis) and vertical (y-axis) directions.

Based on this error, the client sends an HTTP command (e.g., /action?go=left) to the ESP32-CAM server. The ESP32-CAM receives the command and activates the corresponding servo motor(s) to rotate the camera toward the object.

This process repeats in a closed-loop cycle: Capture → Process → Decide → Act until the object is centered in the frame.

III .4.4.4 Step 4: Tracking Confirmation (Visual Feedback)

Once the object is detected and centered, the system marks the target with a green dot overlaid on the live video feed. This green dot represents the centroid of the detected blue object and confirms that the tracking algorithm is functioning correctly.

The camera continues to make real-time adjustments to keep the green dot near the center of the frame, ensuring stable tracking even if the object moves.

The experimental results clearly demonstrate the functionality of the ESP32-CAM-based tracking system. Before detection, the system continuously scanned its environment without identifying any target. Upon introducing a blue-colored object, the camera successfully detected and tracked it, marking it with a green dot on the live video feed. This confirms that the system effectively achieves its intended goal: real-time tracking of a colored object using embedded vision and servo control.

III .5 Conclusion

This chapter presented the complete design, assembly, and testing of a 3D camera stabilization and tracking system based on the ESP32-CAM module. The hardware section described the integration of the ESP32-CAM with servo motors to enable pan and tilt movements, supported by a custom mechanical frame and powered through appropriate electrical connections. On the software side, image processing algorithms and color-based tracking were implemented using OpenCV and Python to detect and follow colored objects in real time.

The experimental setup successfully demonstrated the system's ability to track a blue object. Before detection, the camera actively searched for targets. Once the blue tube was placed within its field of view, the system identified it and marked it visually with a green dot. This confirmed the system's expected behavior and validated the effectiveness of combining

embedded systems, computer vision, and servo control in a low-cost, real-time tracking solution.



General conclusion

General conclusion

This dissertation has explored the design, implementation, and evaluation of a low-cost camera stabilization and tracking system built around the ESP32-CAM microcontroller. By integrating embedded vision algorithms, servo-based pan-tilt mechanics, and a lightweight software stack, the project demonstrates that real-time object detection and color-based tracking can be achieved within stringent constraints of power, size, and budget. The successful detection and continuous following of a colored target validate both the theoretical foundations in computer vision and the practical feasibility of leveraging resource-limited hardware for sophisticated tasks.

A key contribution of this work lies in showing how the HSV color model, combined with OpenCV routines, can be effectively deployed on an ESP32-CAM platform to maintain robust performance under variable lighting and motion conditions. Equally important is the demonstration of multithreaded Python-based GUI control via HTTP, which provides an intuitive human-machine interface without compromising real-time responsiveness. These elements collectively illustrate a balanced system architecture that meets the dual goals of accuracy and efficiency.

Despite these achievements, the system exhibits some limitations. The restricted memory and processing capacity of the ESP32-CAM impose bounds on image resolution, frame rate, and algorithmic complexity. Environmental factors such as low light or highly reflective surfaces can degrade tracking accuracy, and sustained Wi-Fi connectivity remains a single point of vulnerability in field deployments. Addressing these constraints will require careful optimization at both the hardware (e.g., power regulation, sensor calibration) and software (e.g., lightweight neural networks, adaptive thresholding) levels.

Looking forward, further enhancements could include the integration of depth sensing, either via stereo vision or low-cost LiDAR modules, to extend the system's capabilities from 2.5D to full 3D spatial awareness. Implementing more advanced machine-learning models for object recognition and predictive motion planning could also improve tracking robustness in cluttered or dynamic environments. Finally, migrating the control interface to a mobile application or edge-cloud hybrid framework would expand usability and scale, paving the way for applications in drone-based inspection, automated surveillance, and interactive robotics. This work lays a solid foundation for such future developments, underlining the potential of affordable, embedded vision platforms in next-generation smart systems.

A rectangular label with a decorative, slightly irregular border, filled with a grey and white marbled pattern. The word "References" is centered on the label in a bold, black, serif font.

References

A rectangular area with a black border containing a marbled paper pattern. The pattern consists of swirling, vein-like shapes in shades of gray and white. The word "Appendices" is centered within this area.

Appendices

References

1. J. Vinothkumar and A. Karunamurthy, “Recent advancements in artificial intelligence technology: Trends and implications,” *Quing: Int. J. Multidiscip. Sci. Res. Develop.*, vol. 2, no. 1, pp. 1–11, 2023.
2. A. Biswal, “7 types of artificial intelligence that you should know in 2023,” *Simplilearn*, Apr. 2023. [Online]. Available: <https://www.simplilearn.com/tutorials/artificial-intelligence-tutorial/types-of-artificial-intelligence.html>
3. M. Bekbolatova *et al.*, “Transformative potential of AI in healthcare: Definitions, applications, and navigating the ethical landscape and public perspectives,” *Healthcare*, vol. 12, no. 2, p. 125, Jan. 2024, doi: 10.3390/healthcare12020125.
4. Y. Lu, “Artificial intelligence: A survey on evolution, models, applications and future trends,” *J. Manage. Anal.*, vol. 6, no. 1, pp. 1–29, 2019, doi: 10.1080/23270012.2019.1570365.
5. S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 4th ed. Hoboken, NJ, USA: Pearson, 2021.
6. L. Floridi and J. Cowls, “A unified framework of five principles for AI in society,” *Harvard Data Sci. Rev.*, vol. 1, no. 1, 2019, doi: 10.1162/99608f92.8cd550d1.
7. M. A. Boden, *AI: Its Nature and Future*. Oxford, U.K.: Oxford Univ. Press, 2016.
8. M. I. Jordan and T. M. Mitchell, “Machine learning: Trends, perspectives, and prospects,” *Science*, vol. 349, no. 6245, pp. 255–260, Jul. 2015, doi: 10.1126/science.aaa8415.
9. A. M. Turing, “Computing machinery and intelligence,” *Mind*, vol. 59, no. 236, pp. 433–460, Oct. 1950, doi: 10.1093/mind/LIX.236.433.
10. D. Rastogi, “Choosing between machine learning and deep learning: A guide for decision-makers,” *LinkedIn*, 2024. [Online]. Available: <https://www.linkedin.com/pulse/choosing-between-machine-learning-deep-guide-deepesh-rastogi-ocbtc>
11. S. Chan *et al.*, “Machine learning in dermatology: Current applications, opportunities, and limitations,” *Dermatol. Therapy*, vol. 10, no. 3, pp. 365–386, Jun. 2020, doi: 10.1007/s13555-020-00372-0.

12. I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016.
13. E. Alpaydin, *Introduction to Machine Learning*, 4th ed. Cambridge, MA, USA: MIT Press, 2020.
14. A. Banafa, *Artificial Intelligence and the Future of Humanity*. Gistrup, Denmark: River Publishers, 2021.
15. S. A. Bhat and F. Infascelli, “Artificial intelligence for environmental sustainability: Opportunities,” *Sustainability*, vol. 12, no. 21, p. 8864, Oct. 2020, doi: 10.3390/su12218864.
16. J. Buolamwini and T. Gebru, “Gender shades: Intersectional accuracy disparities in commercial gender classification,” in *Proc. 1st Conf. Fairness, Accountability Transp. (FAT)*, New York, NY, USA, 2018, pp. 77–91.
17. M. Tegmark, *Life 3.0: Being Human in the Age of Artificial Intelligence*. New York, NY, USA: Penguin Books, 2017.
18. Admin, “3 unexpected uses of AI,” *Bounce Mag.*, 2024. [Online]. Available: <https://bouncemagazine.co.uk/3-unexpected-uses-of-ai/>
19. European Commission, “Proposal for a regulation laying down harmonised rules on artificial intelligence (artificial intelligence act) and amending certain union legislative acts,” COM(2021) 206 final, Apr. 2021. [Online]. Available: <https://digital-strategy.ec.europa.eu/en/library/proposal-regulation-laying-down-harmonised-rules-artificial-intelligence>
20. B. Marr, *Artificial Intelligence in Practice: How 50 Successful Companies Used AI and Machine Learning to Solve Problems*. Chichester, U.K.: Wiley, 2018.
21. N. Bostrom, *Superintelligence: Paths, Dangers, Strategies*. Oxford, U.K.: Oxford Univ. Press, 2014.
22. N. J. Nilsson, *The Quest for Artificial Intelligence*. Cambridge, U.K.: Cambridge Univ. Press, 2010.

23. A. Ouafi, "Mapping multi-objectif à l'aide d'un algorithme génétique pour les applications embarquées," M.S. thesis, Univ. Annaba, Annaba, Algeria, 2014. [Online]. Available: <https://www.slideserve.com/muriel/mapping-multi-objectif-l-aide-d-un-algorithme-g-n-tique-pour-les-applications-embarque-es>
24. S. Zuboff, *The Age of Surveillance Capitalism: The Fight for a Human Future at the New Frontier of Power*. New York, NY, USA: PublicAffairs, 2019.
25. M. Portolan, "Conception d'un système embarqué sûr et sécurisé," Ph.D. dissertation, Grenoble INP, Grenoble, France, 2007.
26. F. Singhoff, "Les systèmes embarqués temps réels VxWorks," Univ. Brest, Brest, France, 2003. [Online]. Available: http://beru.univ-brest.fr/~singhoff/ENS/UE_temps_reel/CM/vx.pdf
27. K. Kasen, "Architecture globale," SlideServe, 2014. [Online]. Available: <https://www.slideserve.com/kasen/architecture-globale>
28. R. Mohammed and O. Abdalhak, "Ordonnancement hiérarchique d'application temps réel sur une architecture NOC," M.S. thesis, Univ. Oran, Oran, Algeria, Jun. 2009.
29. PCB HERO, "Embedded systems memory types," *PCB HERO Blog*, Jul. 2024. [Online]. Available: <https://www.pcb-hero.com/blogs/lickys-column/embedded-systems-memory-types>
30. M. Khalether, "Systèmes embarqué-temps réel," M.S. thesis, Univ. Oran, Oran, Algeria, 2012.
31. Anon., "Konspekt," *Martini Portfoolio*, 2023. [Online]. Available: <https://hbeegies.wordpress.com/konspekt-2/>
32. G. Massine, "Implémentation d'un réseau de neurones dans un microcontrôleur," Dept. Electron., Univ. Mouloud Mammeri Tizi-Ouzou, Tizi-Ouzou, Algeria, 2016.
33. Allelco Limited, "Comparison of RP2040, ESP32, and STM32 microcontrollers," *Allelco Blog*, 2024. [Online]. Available: <https://www.allelcoelec.com/blog/comparison-of-rp2040,esp32,and-stm32-microcontrollers.html>
34. Anon., "ESP32-D0WD Wifi," n.d. [Online]. Available: <https://bbs.16rd.com/thread-607606-1-1.html>

35. Anon., “[Alizeker Mall] Esp32 Cam ESP32-CAM ESP-32S WiFi module ESP32 serial to ESP32 CAM 5V Bluetooth with OV2640 camera module,” *Daraz.com.bd*, n.d. [Online]. Available: <https://www.daraz.com.bd/products/alizeker-mallesp32-cam-esp32-cam-esp-32s-wifi-module-esp32-serial-to-esp32-cam-5v-bluetooth-with-ov2640-camera-module-i264927222.html>
36. PPE TS1, “Le Microcontrôleur,” 2025. [Online]. Available: <https://www.ppe-ssl-essuiemain.fr.gd/Le-Microcontr>
37. Anon., “Ports / pins libres sur ESCP32 CAM,” *Arduino Forum*, 2021. [Online]. Available: <https://forum.arduino.cc/t/ports-pins-libres-sur-escp32-cam/881002>
38. PPE TS1, “Le Microcontrôleur,” 2020. [Online]. Available: <https://www.ppe-ssl-essuiemain.fr.gd/Le-Microcontr>
39. Anon., “Wi-Fi API — Arduino ESP32 latest documentation,” *Espressif*, n.d. [Online]. Available: <https://docs.espressif.com/projects/arduino-esp32/en/latest/api/wifi.html?highlight=ftm>
40. R. Boulkroune, “Les systèmes embarqués,” Univ. Annaba, Annaba, Algeria, Jun. 2009.
41. M. Barr and A. Massa, *Programming Embedded Systems: With C and GNU Development Tools*, 2nd ed. Sebastopol, CA, USA: O’Reilly Media, 2006.
42. M. A. Mazidi, J. G. Mazidi, and R. D. McKinlay, *The 8051 Microcontroller and Embedded Systems: Using Assembly and C*, 2nd ed. Upper Saddle River, NJ, USA: Pearson Educ., 2008.
43. Anon., “Problem uploading sketch into ESP32-CAM module!,” *Arduino Forum*, 2023. [Online]. Available: <https://forum.arduino.cc/t/problem-uploading-sketch-into-esp32-cam-module/1167297>
44. Anon., “SG90 9G mini servo motor compatible with RC helicopter, car, boat, arm, robot, walker, and door lock (2) – Precise and versatile servo for your projects,” *Amazon.ae*, n.d. [Online]. Available: <https://www.amazon.com>
45. N. Kolban, *Kolban’s Book on ESP32*. Self-published, 2017.

46. Anon., “SanDisk 32GB micro SDHC memory card UHS speed class 1 / speed class...,” *Fruugo*, n.d. [Online]. Available: <https://www.fruugo.fr/sandisk-32gb-micro-sdhc-memory-card-uhs-speed-class-1-speed-class-10-10mbs-black/p-57850458-117321495?language=en>
47. E. A. Rodríguez, *Getting Started with ESP32-CAM: Build Projects Using Arduino IDE and ESP32-CAM Module*. Maarheize, Netherlands: Elektor Int. Media, 2020.
48. L. Prieto, *Programming the ESP32 with Arduino IDE*. Independently published, 2021.
49. Anon., “Adaptateur USB vers TTL, CH340G,” *MHTronic*, n.d. [Online]. Available: <https://mhtronic.com/produit/adaptateur-usb-vers-ttl-ch340/>
50. Geeks Valley, “Number system converter using Arduino,” *GeeksValley*, 2023. [Online]. Available: <https://geek山谷.com/en/tutorial/number-conversion-by-arduino/?view=all>
51. Espressif Systems, “ESP32-CAM development board with OV2640 camera module [Datasheet],” 2020.
52. IRISA, “Technological survey,” May 2025. [Online]. Available: <http://www.irisa.fr/caps/projects/TechnologicalSurvey>
53. Anon., “Pan tilt servo gimbal for FPV drone board cameras – Includes 2x 9 g servo,” *Ubuy jo*, n.d. [Online]. Available: <https://www.ubuy.com.jo/ar/product/30DKXCS08-pan-tilt-servo-gimbal-for-fpv-drone-board-cameras-includes-2x-9g-servo>
54. Anon., “What is a charger? Exploring the basics,” *Lenovo US*, 2023. [Online]. Available: https://www.lenovo.com/us/en/glossary/what-is-charger/?srsltid=AfmBOooG-PqLAcIfQr6p6b_FEBML127_rpxtC9cK1pnCzZd47Ob2UVu-

Appendices

Arduino code for esp32cam camera and control in 2 Servo Motor

```
#include "esp_camera.h"
#include <WiFi.h>
#include "esp_http_server.h"
#include <ESP32Servo.h>

// إعدادات السيرفو
#define SERVO_PAN_PIN 13
#define SERVO_TILT_PIN 15
Servo servoPan;
Servo servoTilt;
int posPan = 90;
int posTilt = 90;

// إعدادات الكاميرا AI-THINKER
#define PWDN_GPIO_NUM 32
#define RESET_GPIO_NUM -1
#define XCLK_GPIO_NUM 0
#define SIOD_GPIO_NUM 26
#define SIOC_GPIO_NUM 27
#define Y9_GPIO_NUM 35
#define Y8_GPIO_NUM 34
#define Y7_GPIO_NUM 39
#define Y6_GPIO_NUM 36
#define Y5_GPIO_NUM 21
#define Y4_GPIO_NUM 19
#define Y3_GPIO_NUM 18
#define Y2_GPIO_NUM 5
#define VSYNC_GPIO_NUM 25
#define HREF_GPIO_NUM 23
#define PCLK_GPIO_NUM 22
```

```

const char* ssid = "ESP32CAM";
const char* password = "12345678";

// ثابت IP
IPAddress local_IP(192, 168, 137, 10); // ثابت تختاره أنت IP
IPAddress gateway(192, 168, 137, 1); // عنوان البوابة (اللاب توب)
IPAddress subnet(255, 255, 255, 0); // قناع الشبكة

#include <WebServer.h>
WebServer server(80);

httpd_handle_t camera_httpd = NULL;

static esp_err_t index_handler(httpd_req_t *req){
    const char* resp = "<html><body><h1>ESP32-CAM</h1></body></html>";
    httpd_resp_send(req, resp, strlen(resp));
    return ESP_OK;
}

static esp_err_t capture_handler(httpd_req_t *req){
    camera_fb_t * fb = esp_camera_fb_get();
    if (!fb) {
        httpd_resp_send_500(req);
        return ESP_FAIL;
    }
    httpd_resp_set_type(req, "image/jpeg");
    httpd_resp_send(req, (const char *)fb->buf, fb->len);
    esp_camera_fb_return(fb);
    return ESP_OK;
}

static esp_err_t cmd_handler(httpd_req_t *req){

```

```

char buf[100];
size_t buf_len = httpd_req_get_url_query_len(req) + 1;
if (buf_len > 1) {
    if (httpd_req_get_url_query_str(req, buf, buf_len) == ESP_OK) {
        char param[20];
        if (httpd_query_key_value(buf, "go", param, sizeof(param)) == ESP_OK) {
            if (strcmp(param, "left") == 0 && posPan <= 170) {
                posPan += 1; //5
                //servoPan.write(posPan);
            } else if (strcmp(param, "right") == 0 && posPan >= 10) {
                posPan -= 1; //5
                //servoPan.write(posPan);
            } else if (strcmp(param, "up") == 0 && posTilt <= 170) {
                posTilt += 1;
                //servoTilt.write(posTilt);
            } else if (strcmp(param, "down") == 0 && posTilt >= 10) {
                posTilt -= 1;
                //servoTilt.write(posTilt);
            }
            servoPan.write(posPan);
            servoTilt.write(posTilt);
        }
    }
}
httpd_resp_set_hdr(req, "Access-Control-Allow-Origin", "*");
return httpd_resp_send(req, NULL, 0);
}

```

```

static esp_err_t start_handler(httpd_req_t *req){
    // شغل الوميض و عدل الزوايا - start عند استقبال
    servoPan.attach(SERVO_PAN_PIN, 500, 2500);
    servoTilt.attach(SERVO_TILT_PIN, 500, 2500);
    posPan = 90;
    posTilt = 90;
    servoPan.write(posPan);
}

```

```

servoTilt.write(posTilt);

for (int i = 0; i < 5 * 10; i++) { // وميض سريع 5 ثواني
    digitalWrite(4, !digitalRead(4));
    delay(50);
}
digitalWrite(4, LOW);

httpd_resp_set_hdr(req, "Access-Control-Allow-Origin", "*");
return httpd_resp_send(req, "STARTED", HTTPD_RESP_USE_STRLEN);
}

void startCameraServer(){
    httpd_config_t config = HTTPD_DEFAULT_CONFIG();
    config.server_port = 80;

    httpd_uri_t index_uri = { .uri="/", .method=HTTP_GET, .handler=index_handler,
    .user_ctx=NULL };

    httpd_uri_t capture_uri = { .uri="/capture", .method=HTTP_GET,
    .handler=capture_handler, .user_ctx=NULL };

    httpd_uri_t cmd_uri = { .uri="/action", .method=HTTP_GET, .handler=cmd_handler,
    .user_ctx=NULL };

    httpd_uri_t start_uri = { .uri="/start", .method=HTTP_GET, .handler=start_handler,
    .user_ctx=NULL };

    httpd_start(&camera_httpd, &config);
    httpd_register_uri_handler(camera_httpd, &index_uri);
    httpd_register_uri_handler(camera_httpd, &capture_uri);
    httpd_register_uri_handler(camera_httpd, &cmd_uri);
    httpd_register_uri_handler(camera_httpd, &start_uri);
}

void setup() {
    Serial.begin(115200);
    // ثابت قبل الاتصال IP تعيين

```



```

if (!WiFi.config(local_IP, gateway, subnet)) {
    Serial.println("STA Failed to configure");
}

WiFi.begin(ssid, password);

while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}

Serial.println("");
Serial.print("ESP32 IP: ");
Serial.println(WiFi.localIP());

pinMode(4, OUTPUT);
digitalWrite(4, HIGH); // كمؤشر بدء الاتصال LED شغل الـ

// إعداد الكاميرا
camera_config_t config;
config.ledc_channel = LEDC_CHANNEL_0;
config.ledc_timer = LEDC_TIMER_0;
config.pin_d0 = Y2_GPIO_NUM;
config.pin_d1 = Y3_GPIO_NUM;
config.pin_d2 = Y4_GPIO_NUM;
config.pin_d3 = Y5_GPIO_NUM;
config.pin_d4 = Y6_GPIO_NUM;
config.pin_d5 = Y7_GPIO_NUM;
config.pin_d6 = Y8_GPIO_NUM;
config.pin_d7 = Y9_GPIO_NUM;
config.pin_xclk = XCLK_GPIO_NUM;
config.pin_pclk = PCLK_GPIO_NUM;
config.pin_vsync = VSYNC_GPIO_NUM;
config.pin_href = HREF_GPIO_NUM;

```

```

config.pin_sscb_sda = SIOD_GPIO_NUM;
config.pin_sscb_scl = SIOC_GPIO_NUM;
config.pin_pwdn = PWDN_GPIO_NUM;
config.pin_reset = RESET_GPIO_NUM;
config.xclk_freq_hz = 20000000;
config.pixel_format = PIXFORMAT_JPEG;

```

```

if(psramFound()){
    config.frame_size = FRAMESIZE_VGA;
    config.jpeg_quality = 10;
    config.fb_count = 2;
} else {
    config.frame_size = FRAMESIZE_CIF;
    config.jpeg_quality = 12;
    config.fb_count = 1;
}

```

```

if (esp_camera_init(&config) != ESP_OK) {
    Serial.println("Camera init failed");
    return;
}

```

```

// الاتصال بالواي فاي
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}
Serial.println("");
Serial.print("WiFi connected! IP: ");
Serial.println(WiFi.localIP());

```

```

// LED بعد الاتصال – طفي الـ
digitalWrite(4, LOW);

```

```

startCameraServer();
}

void loop() {
    server.handleClient();

}

```

Object color tracking and camera movement code

```

#python "C:\Users\othmn\OneDrive\Desktop\DecetPython\New folder\New
folder\New folder\New folder\color_tracking_gui.py"

import tkinter as tk
import threading
import cv2
import numpy as np
import requests

ESP32_IP = "192.168.137.10"

class App:
    def __init__(self, root):
        self.root = root
        self.root.title("Color Tracking ESP32-CAM")
        self.running = False

        # زر البدء
        self.start_button = tk.Button(root, text="Open to start
tracking", command=self.start_tracking, width=30, height=2)
        self.start_button.pack(pady=10)

        # منطقة عرض الرسائل
        self.msg_label = tk.Label(root, text="", font=("Helvetica", 16))
        self.msg_label.pack(pady=10)

        # زر الإيقاف
        self.stop_button = tk.Button(root, text="Stop",
command=self.stop_tracking, width=15, height=2, state="disabled")
        self.stop_button.pack(pady=10)

        # تحقق الاتصال بعد بدء الواجهة
        self.root.after(100, self.check_connection)

    def check_connection(self):
        try:
            resp = requests.get(f"http://{ESP32_IP}/capture", timeout=2)
            if resp.status_code == 200:
                # لون الزر أخضر داكن والنص أبيض
                self.start_button.config(bg="#228B22", fg="white")
                self.update_msg("ESP32 Connected!")
            else:

```

```

        self.start_button.config(bg="SystemButtonFace",
fg="black")
        self.update_msg("ESP32 Not Connected Yet!")
    except:
        self.start_button.config(bg="SystemButtonFace", fg="black")
        self.update_msg("ESP32 Not Connected Yet!")

    # تحقق كل 3 ثواني من الاتصال
    if not self.running:
        self.root.after(3000, self.check_connection)

def start_tracking(self):
    self.running = True
    self.msg_label.config(text="Tracking started!")
    self.start_button.config(state="disabled")
    self.stop_button.config(state="normal")
    self.thread = threading.Thread(target=self.track)
    self.thread.start()

def stop_tracking(self):
    self.running = False
    self.msg_label.config(text="Tracking stopped.")
    self.start_button.config(state="normal",
fg="white") # اللون الأخضر للنص
    self.stop_button.config(state="disabled")

def track(self):
    requests.get(f"http://{ESP32_IP}/start")
    # اجعل النافذة قابلة للتكبير وممتلئة الشاشة
    cv2.namedWindow("ESP32 Blue Tracking", cv2.WINDOW_NORMAL)
    #cv2.namedWindow("ESP32 Blue Tracking", cv2.WINDOW_NORMAL)
    #cv2.setWindowProperty("ESP32 Blue Tracking",
cv2.WND_PROP_FULLSCREEN, cv2.WINDOW_FULLSCREEN)
    while self.running:
        try:
            resp = requests.get(f"http://{ESP32_IP}/capture",
timeout=5)
            arr = np.frombuffer(resp.content, dtype=np.uint8)
            img = cv2.imdecode(arr, cv2.IMREAD_COLOR)
        except:
            self.update_msg("Failed to detect image!")
            continue

    if img is not None:
        hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
        lower_blue = np.array([100, 150, 0])
        upper_blue = np.array([140, 255, 255])
        mask = cv2.inRange(hsv, lower_blue, upper_blue)

        M = cv2.moments(mask)
        if M["m00"] != 0:
            cx = int(M["m10"] / M["m00"])
            cy = int(M["m01"] / M["m00"])
            cv2.circle(img, (cx, cy), 10, (0, 255, 0), -1)

        width = img.shape[1]
        height = img.shape[0]
        center_x = width // 2
        center_y = height // 2
        error_x = cx - center_x
        error_y = cy - center_y

```

```

        dead_zone = 20

        if abs(error_x) <= dead_zone and abs(error_y) <=
dead_zone:
            self.update_msg("In the Center")
        else:
            if error_x < -dead_zone:
                self.update_msg("Left...")

requests.get(f"http://{ESP32_IP}/action?go=left")
            elif error_x > dead_zone:
                self.update_msg("Right...")

requests.get(f"http://{ESP32_IP}/action?go=right")

            if error_y < dead_zone:
                self.update_msg("Up...")

requests.get(f"http://{ESP32_IP}/action?go=down")
            elif error_y > -dead_zone:
                self.update_msg("Down...")

requests.get(f"http://{ESP32_IP}/action?go=up")
        else:
            self.update_msg("In the Center")

        cv2.imshow("ESP32 Blue Tracking", img)

        #if cv2.waitKey(1) & 0xFF == ord('ESC'):
        if cv2.waitKey(1) & 0xFF == 27: # 27 هو كود زر ESC
            self.stop_tracking()
            break

    else:
        self.update_msg("Failed to detect image!")

    cv2.destroyAllWindows()
    self.update_msg("Tracking stopped.")

    def update_msg(self, text):
        self.msg_label.config(text=text)

root = tk.Tk()
app = App(root)
root.mainloop()

```

More color tracking code and camera movement

```

# python "C:\Users\othmn\OneDrive\Desktop\DecetPython\New folder\New
folder\New folder\New folder (3)\color_tracking_guiBOT0Multi.py"

import tkinter as tk
import threading
import cv2
import numpy as np
import requests

```

```
ESP32_IP = "192.168.137.10"
```

```
class App:
    def __init__(self, root):
        self.root = root
        self.root.title("Color Tracking ESP32-CAM")
        self.running = False
        self.color_to_track = "blue"

        # زر البدء
        self.start_button = tk.Button(root, text="Open to start
tracking", command=self.start_tracking, width=30, height=2,
state="disabled")
        self.start_button.pack(pady=10)

        # أزرار اختيار اللون
        frame = tk.Frame(root)
        frame.pack(pady=10)

        self.blue_button = tk.Button(frame, text="Blue", command=lambda:
self.set_color("blue"), bg="blue", fg="white", width=10)
        self.blue_button.pack(side="left", padx=5)

        self.red_button = tk.Button(frame, text="Red", command=lambda:
self.set_color("red"), bg="red", fg="white", width=10)
        self.red_button.pack(side="left", padx=5)

        self.green_button = tk.Button(frame, text="Green",
command=lambda: self.set_color("green"), bg="green", fg="white",
width=10)
        self.green_button.pack(side="left", padx=5)

        self.yellow_button = tk.Button(frame, text="Yellow",
command=lambda: self.set_color("yellow"), bg="yellow", fg="black",
width=10)
        self.yellow_button.pack(side="left", padx=5)

        # منطقة عرض الرسائل
        self.msg_label = tk.Label(root, text="Connecting to ESP32...",
font=("Helvetica", 16))
        self.msg_label.pack(pady=10)

        # زر الإيقاف
        self.stop_button = tk.Button(root, text="Stop",
command=self.stop_tracking, width=15, height=2, state="disabled")
        self.stop_button.pack(pady=10)

        self.check_connection()
        self.update_color_buttons()

    def set_color(self, color):
        self.color_to_track = color
        self.update_color_buttons()
        self.update_msg(f"Tracking color: {color.capitalize()}")

    def update_color_buttons(self):
        # الألوان الأصلية
        self.blue_button.config(bg="blue", fg="white")
        self.red_button.config(bg="red", fg="white")
        self.green_button.config(bg="green", fg="white")
        self.yellow_button.config(bg="yellow", fg="black")
```

```

# لون خفيف جدًا للزر المختار
if self.color_to_track == "blue":
    self.blue_button.config(bg="#ADD8E6")
elif self.color_to_track == "red":
    self.red_button.config(bg="#FFB6C1")
elif self.color_to_track == "green":
    self.green_button.config(bg="#90EE90")
else:
    self.yellow_button.config(bg="#FFFF99") # أصفر فاتح جدًا

def check_connection(self):
    try:
        resp = requests.get(f"http://{ESP32_IP}/capture", timeout=2)
        if resp.status_code == 200:
            self.start_button.config(bg="#228B22", fg="white",
state="normal")
            self.update_msg("ESP32 Connected!")
            return
        else:
            self.start_button.config(bg="SystemButtonFace",
fg="black", state="disabled")
    except:
        self.start_button.config(bg="SystemButtonFace", fg="black",
state="disabled")

    self.update_msg("Connecting to ESP32...")
    if not self.running:
        self.root.after(3000, self.check_connection)

def start_tracking(self):
    self.running = True
    self.msg_label.config(text="Tracking started!")
    self.start_button.config(state="disabled")
    self.stop_button.config(state="normal")
    self.thread = threading.Thread(target=self.track)
    self.thread.start()

def stop_tracking(self):
    self.running = False
    self.msg_label.config(text="Tracking stopped.")
    self.start_button.config(state="disabled",
bg="SystemButtonFace", fg="black")
    self.stop_button.config(state="disabled")
    self.root.after(100, self.check_connection)

def track(self):
    requests.get(f"http://{ESP32_IP}/start")
    cv2.namedWindow("ESP32 Color Tracking", cv2.WINDOW_NORMAL)
    while self.running:
        try:
            resp = requests.get(f"http://{ESP32_IP}/capture",
timeout=5)
            arr = np.frombuffer(resp.content, dtype=np.uint8)
            img = cv2.imdecode(arr, cv2.IMREAD_COLOR)
        except:
            self.update_msg("Failed to detect image!")
            continue

        if img is not None:
            hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

```

```

        if self.color_to_track == "blue":
            lower = np.array([100, 150, 30])
            upper = np.array([140, 255, 100])
        elif self.color_to_track == "red":
            lower = np.array([0, 150, 50])
            upper = np.array([10, 255, 255])
        elif self.color_to_track == "green":
            lower = np.array([40, 70, 50])
            upper = np.array([80, 255, 255])
        else: # yellow
            lower = np.array([20, 100, 100])
            upper = np.array([30, 255, 255])

        mask = cv2.inRange(hsv, lower, upper)

        M = cv2.moments(mask)
        if M["m00"] != 0:
            cx = int(M["m10"] / M["m00"])
            cy = int(M["m01"] / M["m00"])
            cv2.circle(img, (cx, cy), 10, (0, 255, 0), -1)

            width = img.shape[1]
            height = img.shape[0]
            center_x = width // 2
            center_y = height // 2
            error_x = cx - center_x
            error_y = cy - center_y
            dead_zone = 40

            if abs(error_x) <= dead_zone and abs(error_y) <=
dead_zone:
                self.update_msg("In the Center")
            else:
                if error_x < -dead_zone:
                    self.update_msg("Left...")

requests.get(f"http://{ESP32_IP}/action?go=left")
                elif error_x > dead_zone:
                    self.update_msg("Right...")

requests.get(f"http://{ESP32_IP}/action?go=right")

                if error_y < dead_zone:
                    self.update_msg("Up...")

requests.get(f"http://{ESP32_IP}/action?go=down")
                elif error_y > -dead_zone:
                    self.update_msg("Down...")

requests.get(f"http://{ESP32_IP}/action?go=up")
            else:
                self.update_msg("WAITING")

            cv2.imshow("ESP32 Color Tracking", img)
            if cv2.waitKey(1) & 0xFF == 27: # ESC key
                self.stop_tracking()
                break
        else:
            self.update_msg("Failed to detect image!")

```



```

cv2.destroyAllWindows()
self.update_msg("Tracking stopped.")

def update_msg(self, text):
    self.msg_label.config(text=text)

root = tk.Tk()
app = App(root)
root.mainloop()

```

Code that follows the face

```

# python "C:\Users\othmn\OneDrive\Desktop\New folder
(11)\face_tracking_esp32_gui.py"

import tkinter as tk
import threading
import cv2
import numpy as np
import requests

ESP32_IP = "192.168.137.10"

# لوجوه DNN تحميل نموذج
configFile = "deploy.prototxt"
modelFile = "res10_300x300_ssd_iter_140000.caffemodel"
net = cv2.dnn.readNetFromCaffe(configFile, modelFile)

class App:
    def __init__(self, root):
        self.root = root
        self.root.title("ESP32 Face Tracking")
        self.running = False

        # زر البدء
        self.start_button = tk.Button(root, text="Start Tracking",
command=self.start_tracking, width=30, height=2, state="disabled")
        self.start_button.pack(pady=10)

        # منطقة عرض الرسائل
        self.msg_label = tk.Label(root, text="Connecting to ESP32...",
font=("Helvetica", 16))
        self.msg_label.pack(pady=10)

        # زر الإيقاف
        self.stop_button = tk.Button(root, text="Stop",
command=self.stop_tracking, width=15, height=2, state="disabled")
        self.stop_button.pack(pady=10)

        self.check_connection()

    def check_connection(self):
        try:
            resp = requests.get(f"http://{ESP32_IP}/capture", timeout=2)
            if resp.status_code == 200:
                self.start_button.config(bg="#228B22", fg="white",
state="normal")
                self.update_msg("ESP32 Connected!")
            return

```

```

        else:
            self.start_button.config(bg="SystemButtonFace",
fg="black", state="disabled")
        except:
            self.start_button.config(bg="SystemButtonFace", fg="black",
state="disabled")

        self.update_msg("Connecting to ESP32...")
        if not self.running:
            self.root.after(3000, self.check_connection)

    def start_tracking(self):
        self.running = True
        self.msg_label.config(text="Tracking started!")
        self.start_button.config(state="disabled")
        self.stop_button.config(state="normal")
        self.thread = threading.Thread(target=self.track)
        self.thread.start()

    def stop_tracking(self):
        self.running = False
        self.msg_label.config(text="Tracking stopped.")
        self.start_button.config(state="disabled",
bg="SystemButtonFace", fg="black")
        self.stop_button.config(state="disabled")
        self.root.after(100, self.check_connection)

    def track(self):
        requests.get(f"http://{ESP32_IP}/start")
        cv2.namedWindow("ESP32 Face Tracking", cv2.WINDOW_NORMAL)
        while self.running:
            try:
                resp = requests.get(f"http://{ESP32_IP}/capture",
timeout=5)
                arr = np.frombuffer(resp.content, dtype=np.uint8)
                frame = cv2.imdecode(arr, cv2.IMREAD_COLOR)
            except:
                self.update_msg("Failed to get image!")
                continue

            if frame is not None:
                h, w = frame.shape[:2]
                blob = cv2.dnn.blobFromImage(cv2.resize(frame, (300,
300)), 1.0,
                                                (300, 300), (104.0, 177.0,
123.0))

                net.setInput(blob)
                detections = net.forward()

                found = False
                for i in range(detections.shape[2]):
                    confidence = detections[0, 0, i, 2]
                    if confidence > 0.2:
                        found = True
                        box = detections[0, 0, i, 3:7] * np.array([w, h,
w, h])

                        (x1, y1, x2, y2) = box.astype("int")
                        cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255,
0), 2)

                # تتبع المركز

```

```

        cx = (x1 + x2) // 2
        cy = (y1 + y2) // 2
        center_x = w // 2
        center_y = h // 2
        error_x = cx - center_x
        error_y = cy - center_y
        dead_zone = 30

        if abs(error_x) <= dead_zone and abs(error_y) <=
dead_zone:
            self.update_msg("Face in Center")
        else:
            if error_x < -dead_zone:
                self.update_msg("Left...")

requests.get(f"http://{ESP32_IP}/action?go=left")
            elif error_x > dead_zone:
                self.update_msg("Right...")

requests.get(f"http://{ESP32_IP}/action?go=right")

            if error_y < -dead_zone:
                self.update_msg("Up...")

requests.get(f"http://{ESP32_IP}/action?go=down")
            elif error_y > dead_zone:
                self.update_msg("Down...")

requests.get(f"http://{ESP32_IP}/action?go=up")
        break

        if not found:
            self.update_msg("No face detected")

        cv2.imshow("ESP32 Face Tracking", frame)
        if cv2.waitKey(1) & 0xFF == 27:
            self.stop_tracking()
            break
    else:
        self.update_msg("Failed to get image!")

    cv2.destroyAllWindows()
    self.update_msg("Tracking stopped.")

def update_msg(self, text):
    self.msg_label.config(text=text)

root = tk.Tk()
app = App(root)
root.mainloop()

```