A blue square with white text

Description automatically generated

**UNIVERSITY OF HERTFORDSHIRE**

SCHOOL OF PHYSIC, ENGINEERING AND COMPUTER SCIENCE

**MSC Computer Science**

7COM1039-0509-2022 - Advanced Computer Science Master Project

09/2023

**FINAL PROJECT REPORT**

PROJECT TITLE: APPLYING GESTURE DETECTION OF FRAMEWORK MEDIAPIPE AND EMBEDDED COMPUTER TO CONTROLLING ROBOT

|  |  |
| --- | --- |
| Student name:  Student ID:  Supervisor: | Bui Tien Hung  21086152  Dr. Raghubir Singh |

**ABSTRACT**

The primary objective of this project is to utilise the Mediapipe framework, a development of Google, for the purpose of robot control. The robot does many motions, including lateral movement to the left, right, and forward. The robot is comprised of two primary elements, namely the tracked robot chassis and the embedded computer Raspberry Pi 4. The aforementioned chassis is equipped with motors and a crawler vehicle, enabling it to move forward, right-turn, and left-turn movements by transmitting control signals from Raspberry Pi to the motor control board. The purpose of this board is to control the movement of two motors, enabling them to rotate either in a clockwise or anticlockwise direction. The latter consists of an embedded computer system and a camera that is interconnected with the computer. The camera captures images of the hands and afterwards transmits them to the computer. The detection of hand movements is facilitated by a computer application that utilises the Mediapipe framework. Subsequently, the application processes these gestures and generates corresponding high and low signals on the General Purpose Input/Output (GPIO) pins of the computer. Subsequently, the control motors circuit is responsible for receiving signals that control the movement of the motors on the crawler vehicle enabling the vehicle to be controlled following certain gestures. Furthermore, this article serves to supplement two concepts: firstly, the robots are operated from a remote location, and secondly, the robots' cameras possess the capability to direct their attention towards the hand of the operator when it detects the hand.

**MSC FINAL PROJECT DECLARATION**

This report is submitted in partial fulfilment of the requirement for the degree of Master of Science in Advanced Computer Science Masters Project at the University of Hertfordshire (UH). It is my own work except where indicated in the report.

I did not use human participants in my MSc Project.

I hereby give permission for the report to be made available on the university website provided the source is acknowledged.

**TABLE OF CONTENT**

***Abstract.***

***Table of contents.***

***List of Figures.***

**Chapter 1. INTRODUCTION**

* 1. Introduction.
  2. Aim and Projective.
  3. Research question.

**Chapter 2. LITERATURE REVIEW**

2.1. Introduction.

2.2. Using physical components methods.

2.3. Image processing methods.

2.4. Summary.

**Chapter 3. BASIC THEORY, PROGRAMING LANGUAGE AND LIBRARIES**

3.1. Digital image representation.

3.2. Linear algebra (matrix).

3.3. Programming language.

3.4. Mediapipe.

3.5. NumPy library.

3.6. OpenCV library.

3.7. GPIO Zero library

**Chapter 4: DESIGN SYSTEM AND METHODOLOGY**

4.1. Hardware.

4.1.1. Embedded computer Raspberry Pi.

4.1.2. Raspberry Pi Camera.

4.1.3. Motor Driver L293N.

4.1.4. Servo SG90.

4.2. Design system.

4.2.1. design description.

4.2.2. Control Methodology.

**Chapter 5: DEPLOYMENT AND RESULT**

5.1. Process of deployment:

5.1.1. Wiring Diagram.

5.1.2. Inclined angles and focusing process.

5.2. Result.

**Chapter 6: CONCLUSION AND FUTURE WORK**

**LIST OF FIGURES**

Figure 1: Block Diagram of gesture detection block.

Figure 2: Block Diagram of implementation block.

Figure 4: Block Diagram of Receiver.

Figure 5: Block Diagram

Figure 6: The realization of gesture-based control.

Figure 7: Co-ordinations and Data Structure of a Digital Image.

Figure 8: Rate of questions of programming languages

Figure 9: Matrix A mxn with aij belonging to the set of real numbers.

Figure 10: Hand-Knuckles ID diagram.

Figure 11: Pure Python and NumPy are performed on Google Colab.

Figure 12: Raspberry Pi 4 model B board.

Figure 13: Screenshot of Raspberry Pi OS.

Figure 14: GPIOs Diagram of Raspberry Pi 4.

Figure 15: Raspberry Pi Camera.

Figure 16: Motor Driver L298N Circuit.

Figure 17: Servo SG90 Dimension and Specification.

Figure 18: Blocks Diagram.

Figure 19: Flow Process Diagram.

Figure 20: Four Controlling Gestures.

Figure 21: Dividing Control Angles.

Figure 22: Wiring Diagram.

Figure 23: Wire Connection and States Table of Camera Control

Figure 24: Wire Connection and States Table of Crawler Control.

Figure 25: Illustration of Index Finger Inclined Angle.

Figure 26: Conditions of Movements Table.

Figure 27: Region of Wrist Coordinates on Image Frame.

Figure 28: Commands Generated by Hand Gestures.

**CHAPTER I: INTRODUCTION**

**1.1. Introduction:**

The advancements in science and technology have led to significant improvements in several aspects of human existence through the emergence of the robotic industry in the contemporary period. Meeting robots that serve various businesses, such as the automobile industry, household appliances, and agriculture, is a relatively accessible endeavour ([Sharkawy, A.N., 2021](#Ref1)). This article will provide a brief overview of the historical development of the discipline of robotics in order to provide a foundational understanding of robotic principles.

In the history of robotics section, the description of robot and the term “Robotics” were published in 1942 by a Russian-American writer named Isaac Asimov in his short story *“Runabout”*, However, the word “robot” was mentioned for the first time in 1921 by Karen Carpek in his play named *“Rossum’s Universal Robots”* ([Irati *et al*. 2017](#Ref2))*.* 40 years after, the term “robot” was born, In 1961, the first simple robot was manufactured and sold by Unimation company ([Kin-Huat L., 2007](#Ref3)) then their development has continued to be present and applied in huge areas. At the moment, modern robots are complex machines that are combined by various scientific sections such as Computer Science, Mechanical Engineering, and Electrical Engineering, accordingly, the controlling parts of robots are also varied ([Irati *et al*. 2017](#Ref2)).

In the history of robotics section, the introduction of the concept of a robot and the term "Robotics" were released in 1942, as documented in the short story "Runabout" authored by Isaac Asimov, a Russian-American writer. The term "robot" was initially mentioned in 1921 by Karen Carpek in his play titled *"Rossum's Universal Robots"* ([Irati *et al*. 2017](#Ref2)). In 1961, four decades after the coining of the name "robot", the Unimation company successfully constructed and commercially distributed the first simple robot ([Kin-Huat L., 2007](#Ref3)). Since then, the advancement and utilisation of robots have persisted throughout several domains. Currently, contemporary robots are implicated devices that integrate several scientific disciplines, including Computer Science, Mechanical Engineering, and Electrical Engineering. Consequently, the control components of robots exhibit a diverse range ([Irati *et al*. 2017](#Ref2)).

The advancement of technology has enabled the implementation of many control mechanisms for robots, including Radio Frequency (RF), switches, voice commands, and control gloves equipped with integrated sensors. In contemporary times, the advancement of Artificial Intelligence (AI) has facilitated the ability to manage robots by gestures, as indicated by Lavanya ([2017](#Ref4)), employing computer vision techniques. This study presents a methodology for utilising advancements in artificial intelligence (AI) within the domain of action detection to facilitate the recognition and interpretation of hand gestures for the purpose of controlling a robotic system. Furthermore, the camera of the remotely operated robot possesses the capability to autonomously adjust its focus on the hand of the operator, while the robots themselves may be commanded from a specific distance.

**1.2. Aim and Objectives:**

**Aims**

The aims of this project:

* The aim of this study is to examine the advancements in artificial intelligence (AI) and their implications for the development of applications aimed at controlling robotic systems.
* The use of controller programming in the environment of embedded computers.
* Applying knowledge learned from the subjects of the course to solve issues of the project.

**Objectives**

* Throughout the project, mastering basic knowledge about applying techniques of AI’s advancements.
* Developing a functional product.
* Creating a robot that can be controlled by hand gestures. This is the premise for researching AI and robotics in the future.

**1.3. Research questions:**

- Are the hand gestures able to control a robot?

- How do the hand gestures control a robot?

- How to use a camera to detect the gestures of the hands to control a robot?

**CHAPTER 2: LITERATURE REVIEW**

**2.1. Introduction:**

In this section of this article, some methods of controlling robots will be reviewed comprehensively, they are namely the controlling methods by [physical components](#gyro) and [image processing](#Image). All of them are based on hand gestures to control robots however they use different techniques to conduct. The former uses physical devices such as gyro-accelerometers, and sensors that are fitted on the hand to measure movements of the hand and then transmit to the robot to control. The latter differs from the former, a camera is used to record the manipulation of the hand, and then an application processes these images to recognize the commands. Two methods will be described in detail to analyse the advantages and disadvantages of them.

**2.2. Using physical components methods:**

Following to the article was titled *“Gyro-Accelerometer based control of a robotic Arm using AVR Microcontroller”*, which was released in 2014 by Ariful Islam Bhuyan, and Tuton Chandra Mallick. In that, to control a robot arm, authors created a system consisting of two parts, respectively, gesture detection block, and implementation block. The former contained a module MPU-6050, a radio frequency (RF) transmitter Xbee, combined with a centre processing board named Arduino Uno, all of them were supplied energy by an independent power. MPU-6050 is a complex module that includes a 3-axis gyroscope and a 3-axis accelerometer, it was mounted on the top of the hand glove. In addition, an accelerometer sensor (Grab sensor) is placed on the metacarpal portion. Two modules measured the rotation angle and the acceleration movements of the hand and then sent the measured data to the centre processing board to calculate. After being processed by a program that was installed on the centre processing board, data was transmitted to the implementation block via the transmitter Xbee module ([Ariful I. B., 2014](#Ref5)). The latter block included a main board Arduino Mega, a radio frequency receiver Xbee, servos, and they are supported by a separate power. The receiver module of this block received the signals from the transmitter of the gestures detection block and then puts them into the main board. Next, the main board, via specific installed applications, commanded servos to control the motions of the robot arm. The programs and applications that were installed on the main board were written by authors rather than the main board manufacturers. The robot arm executed all the tasks following the initially given desires.



Figure 1: Block Diagram of gesture detection block. ([Ariful I. B., 2014](#Ref5))

A diagram of a computer component

Description automatically generated

Figure 2: Block Diagram of implementation block ([Ariful I. B., 2014](#Ref5))

In 2019, a similar solution was described in another article titled “Hand gesture-controlled Robots” written by Pranay Iyer, Sanjana Tarekar, and Swati Dixit that was published on the IEEE webpage. Their solution was used to control a car robot. They designed the controller, which also contained two modules called Transmitter and Receiver. The first module consisted of an accelerometer ADXL355, an Arduino Lylipad board that integrated an ATmega168 microcontroller, an encoding HT12E, and a wireless block RF433. This module performed the task of recording hand gestures and transmitting signals to the Receiver module via RF433[[6]](#Ref6). The second module included wireless block RF433, decoding HT12D, and motor driver L293D. This solution gave out a working principle including several stages. Firstly, accelerometer ADXL355 recorded the movements of the hand, next, it transmitted data about the position of the hand to Arduino, then this data is encoded and then sent to the RF transmitter. The receiver module, via RF 433 receiver receives data and then sent to the decoder module, after decoding, data is passed to the Motor driver to control the action of robots (Pranay I. et al., 2019).

A blue rectangles with a arrow pointing to a blue rectangle

Description automatically generated

Figure 3: Block Diagram of Transmitter ([Pranay I. et al., 2019](#Ref6))

A blue square with white text

Description automatically generated

Figure 4: Block Diagram of Receiver ([Pranay I. et al., 2019](#Ref6))

**2.3. Image processing methods:**

According to Chanhan G. and his colleagues who were joint authors of an article named “Gestures based Wireless Robotic Control using Image Processing” published on the IEEE webpage in 2015, they could use hand gestures to control robots via image processing [[7]](#Ref7). In this article, the researchers used a model that divided into two units, they were namely the Controlling Unit and Assembly Unit. The first unit included the devices namely a webcam, a PC, a Max232(RS232 to TTL converter), and a Zigbee Transmitter. The second unit contained a Zigbee Receiver, an ATmega16 microcontroller, and a Motor driver (L293D), and two DC motors [[7]](#Ref7). In addition, their method also used MATLAB software combined with Image Acquisition Toolbox to process images. The block diagram of them is illustrated in Figure 4.

A diagram of a machine

Description automatically generated

Figure 5: Block Diagram ([Chanhan G. et al, 2015](#Ref7))

The working principle of this solution was using a webcam to capture the images of the hand and then transmit them to the computer where the frames of video were processed to create static images. These images continued to be processed via MATLAB and Tools which changed them to binary form. Next step, a program, that coded authors, would recognise fingers on images and count them. The number of counted fingers corresponded to the commands in the database of the program, and then the commands were sent to the Assembly Unit via Zigbee Transmitter. The Zigbee receiver of the Assembly unit received these commands and then sent them to the ATmega16 microcontroller. In the last step, ATmega16 received commands and then ordered the Motor to run according to requirements.

In 2017, an interesting article was written by four authors H. Zhao, J. Hu, Y. Zhang, and H. Cheng [[9]](#Ref9), titled *“Hand gesture-based control strategy for mobile robots”*. This article presented entirely the details of the technique to process images of hands including image processing theory and their experimental results. The controlling method of them was based on recognising morphological hand to create the command via a software application. The hand images were captured by a camera, next stage, the Gesture Recognition System would process images and then rend the commands. Next, these commands were sent to robots via a wireless module, and at last, robots performed them. The commands were generated by finger count.

A diagram of a flowchart

Description automatically generated

Figure 6: The realization of gesture-based control. ([Zhao H. et al., 2017](#Ref9))

**2.4. Summary:**

These methods are great ideas for controlling robots by hand gesture detection, they had similar solutions such as using two independent modules and connecting each other between them via wireless technology. However, they belong to two different methods namely using physical components and image processing to recognise gestures of the hands. The first, the physical components method, the computational complexity of them was simple because signals were transmitted directly between the sensors and processing module. However, in cases of elderly or physically disabled people, who cannot wear gloves by themselves, this will bring inconveniences. Moreover, the gloves are integrated with power, sensors, and processing modules that have a certain weight, which causes rigid motion for users. Therefore, these ideas are feasible solutions for regular people but difficult to apply to disabilities. These solutions are similar in idea and method; however, they used different physical components because they were created to control different types of robots. Second, the image processing methods were based on video processing techniques via the process of each frame that was separated from the video. As a result, they overcame the disadvantage of mounting the devices on the hands. However, the control modules were not integrated into the robots, therefore the systems were inconvenient to use. In addition, the image processing solutions used a fixed camera to connect to the computer, this made the robots less flexible in operation. For these reasons, this project will focus on research to carry out the solution inherited from existing solutions. Therefore, this research is based on image processing theory and metric algorithms to surmount the disadvantages.

**CHAPTER 3: BASIC THEORY, PROGRAMING LANGUAGE AND LIBRARIES**

**3.1. Digital image representation:**

Since the first digital computer was developed in the 1940s then the commercial version was sold in the 1950s, it has only been able to deal with two digits “0” and “1” ([G. O’Regan, 2016](#Ref10)), therefore, to process an image by a computer, it needs to be represented image by a suitable data type ([Milan S., et al, 2015](#Ref11)). Because of that, the images have been mathematically represented by a continuous function f(x,y), where two arguments x and y are two co-ordinations in two-dimensional space. When digitalising an image, it means that all the values of f(x,y) are archived in a matrix that has P rows and Q columns with P, Q belonging to the set of natural numbers N. This also means that x and y correspond to P and Q. Consequently, a digital picture has a data structure to represent on the computers. The image in two dimensions space is a rectangle net, they are called grids and each grid contains a value of function f(x,y). Moreover, the product of P times Q is called the resolution of the image (number of pixels).

A graph of a function

Description automatically generated

Figure 7: Co-ordinations and Data Structure of a Digital Image ([Milan S., et al, 2015](#Ref11)).

From the digitisation image to a matrix, Humans can fully access every pixel of a picture to view or calibrate by using mathematical operations on the matrix.

**3.2. Linear algebra (matrix):**

Matrix has a significant role in linear algebra, it contains all elements a­­­ij with *i* belonging to set from 1 to *m*, *j* belonging to set from 1 to *n*, and *n,m* belonging to set natural number (N) ([*Marc P. D. et al., 2020*](#Ref12)). A matric *‘A’* dimension *m*x*n* has a rectangular schema including *m* rows and *n* columns.

A group of black text

Description automatically generated

Figure 8: Matrix A mxn with aij belonging to the set of real numbers ([*Marc P. D. et al., 2020*](#Ref12)).

Matrix has the operations such as addition, multiplication, and some different operations. Nevertheless, in the scope of this project, the manipulations on index *i*, and *j* are used to access pixel positions of pictures, therefore, only the definition of the matrix is discussed to apply to the use of the library NumPy.

**3.3. Programming language:**

Python is a high-level programming language that was released for the first time in 1991 by [Guido van Rossum](https://en.wikipedia.org/wiki/Guido_van_Rossum). This language can be implemented on vast platforms of operating systems such as Windows, Linux, MacOS, and Android. In addition, after 32 years of development, Python has applicability in huge domains of information technology, for instance building a website, applications, and data analysis. Because of that, Python has been used by big technology firms for example Facebook (Meta), Google, and Netflix. Following the website Stack Overflow ([Stack Overflow](#Ref13)], From 2019, it has taken the leading position with approximately 12% of questions )bout programming languages across the world (see Figure 7) and has been keeping that position to the present.

A graph of different colored lines

Description automatically generated

Figure 9: Rate of questions of programming languages ([Stack Overflow](#Ref13))

The first advantage of Python programming language is the simplicity and brevity of statement structure. Python code is believed to be only one-third compared to equivalent languages such as C/C++ or Java ([Mark Lutz, 2009](#Ref14)). It means that typing less typing, less debugging, the result is the efficiency of programmers’ work increased. To illustrate this clearly, consider the classical example named ‘Hello World’ written in C language and Python. In C language, the coder must type four code lines corresponding to sixty-two letters whereas Python needs only one line equivalent to twenty-one letters including spaces.

C code:

#include <stdio.h>

int main() {

printf("Hello World!");

return 0;

}

Python code:

print("Hello World!")

Second, besides its libraries, Python is supported by a huge number of third-party libraries in different tasks such as NumPy, OpenCV, Polar, and Pandas. Indeed, today, Python can call the libraries of other languages such as C/C++, Java, or .NET. These libraries help developers to reduce code lines, speed up the programs, and make the programs clearer.

Finally, cause of the possibility of integrating with other languages, Python supports developers in deploying products simply in a shorter time. In addition, when the programs are shorter, developers control bugs more easily.

Summarization, the advantages of Python for developers cannot be listed. Because of that, Python is more interesting, more popular and supported by more contributors.

**3.4. Mediapipe:**

MediaPipe is an open-source framework developed by one of the greatest in the world firms Google, that provides tools and libraries permitting developers to directly embed detection solutions in applications, such as object detection, hand gesture detection, and image classification ([Google Mediapipe](#Ref15)). It was created to simplify the process of creating multimedia pipelines and processing pipelines for various platforms, including desktop, mobile, and embedded systems. The core purpose of Mediapipe is to offer a streamlined way to develop and deploy applications that require understanding and interpreting visual and audio data. It provides pre-built modules and components that handle tasks like face detection, hand tracking, pose estimation, objection (object tracking in 3D), and more. These modules can be easily integrated into developers’ projects to save time and effort in developing these functionalities from scratch. To apply to this project, one of the vision tasks was used, Hand Landmark Detection.

To create this Hand Landmark Detection, the developers group trained the model with nearly 30 thousand hand pictures, that were captured on vast backgrounds. The Hand Landmark detects 21 points corresponding to 21 Knuckles of the hand (see Figure 8).

A screenshot of a computer

Description automatically generated

Figure 10: Hand-Knuckles ID diagram ([Google Mediapipe](#Ref15))

In addition, the 21 coordinates of these points are contained in an accessible tuple. Based on these ordinates, they can be programmed to execute the requirements of applications.

Consumption, Mediapipe is a library that has a variety of detecting applications. It can be used to create controlling software for robots or read the sign language of mutes.

**3.5. NumPy library:**

According to Ivan Idris, the author of the book titled *“Learning NumPy Array”*, NumPy originated as a part of SciPy and then it became an independent library. The name NumPy was formed from two words Numerical and Python ([Ivan Idris, 2014](#Ref16)), furthermore, it is an open-source library. NumPy has been considered a powerful mathematical library of Python programming language based on its efficient support for operations on arrays and matrices. In addition, NumPy provides shorter, cleaner, and higher executing speed than pure Python code.

First, NumPy provides highly efficient array operations and mathematical functions that are optimized for performance. These operations are implemented in C, making them much faster than equivalent operations performed using standard Python lists. The core feature of NumPy is the ndarray object, which allows you to efficiently store and manipulate arrays of numbers. These arrays can have any number of dimensions, making them suitable for representing various types of data, such as images, signals, and time series.

Second, NumPy code can often be shorter and more concise, it's important to note that this conciseness comes from NumPy's ability to perform operations on entire arrays at once. Moreover, NumPy encourages a vectorized programming style, where operations are performed elementwise on arrays, eliminating the need for explicit loops. This leads to more concise and readable code while taking advantage of optimized low-level implementations. To make this clear, it will be illustrated by a multiplication between Scala and a vector using pure Python and NumPy. Pure Python must use a for loop to call each element of the data array to times to Scala 2 whereas NumPy only use an assignment *data \*=2*.

A screenshot of a computer

Description automatically generated

Figure 11: Pure Python and NumPy are performed on Google Colab (Screenshot).

Third, most NumPy library is written in the C programming language, therefore, the speed of computation is faster.

NumPy is a benefit library of Python language, it supports developers powerfully through its huge number of advantages. These increase computing effectiveness in using Python and make applications that are written by Python run faster.

**3.6. OpenCV library:**

OpenCV stands for "Open-Source Computer Vision Library," and is a prevalent open-source software library utilised for computer vision and machine learning applications. The primary objective of its creation was to build a unified framework for computer vision applications and streamline the incorporation of machine perception into commercial applications ([OpenCV Organisation](#Ref19)). Moreover, OpenCV has been specifically developed to offer an extensive array of tools and functionalities catering to a wide range of computer vision applications. These applications span from fundamental image processing operations to complicated tasks such as object detection, facial recognition, and augmented reality. The OpenCV library was initially created by Intel in the year 1999 and has subsequently undergone consistent contributions through the collaborative efforts of the developer community. OpenCV possesses a wide range of important characteristics that are very relevant and influential in the domains of Machine Learning and Computer Vision.

One notable feature of OpenCV is its status as an open-source library, which has been collaboratively developed by a community of developers. This enables programmers to utilise it for educational purposes, science research, and the development of products without incurring any financial expenses. Furthermore, OpenCV provides support for a wide range of computer languages, including C/C++, Python, and Java, as well as many operating systems, such as Linux, Windows, and MacOS. This facilitates the deployment of apps for developers in a manner that is both efficient and comfortable. OpenCV encompasses a comprehensive collection of over 2,500 optimised algorithms, encompassing both classical and machine learning algorithms. This extensive repertoire renders OpenCV highly resilient and well-suited for utilisation in the domain of image processing.

A noteworthy characteristic of OpenCV is its designation as an open-source library, which has been collaboratively developed by a community of developers. This enables programmers to use it for educational purposes, scientific research, and the development of products without incurring any financial costs. In addition, OpenCV provides support for a wide range of programming languages, including C/C++, Python, and Java, along with compatibility with multiple operating systems, including Linux, Windows, and MacOS. This enables developers to deploy applications in a manner that is efficient and comfortable. OpenCV has an extensive assemblage of more than 2,500 optimised algorithms, including a wide range of classical and machine learning algorithms. The wide range of capabilities possessed by OpenCV makes it extremely adaptable and suitable for applications in the field of image processing.

OpenCV has been widely utilised in numerous fields, including but not limited to robotics, automotive systems, medical imaging, security and surveillance, and entertainment. The tool's flexibility and extensive range of features render it an indispensable asset for anyone engaged in the field of computer vision and its associated disciplines.

**3.7. GPIO Zero library:**

The GPIO Zero library is a Python-based tool designed to facilitate the manipulation of GPIO (General Purpose Input/Output) pins on single-board computers, such as the Raspberry Pi. The tool offers a straightforward and user-friendly approach to interacting with GPIO pins, facilitating the manipulation of hardware elements and the development of projects involving physical computing for individuals of all skill levels, including novices and experts. One of the benefits of using GPIO Zero is its capability to express components of Object Oriented Programming, such as Buttons, LED, and servos, as Python objects. Furthermore, GPIO Zero offers a variety of pre-existing classes for commonly used components. Additionally, it is designed to be expandable, enabling users to create their own personalised components and seamlessly include them into their undertakings.

The GPIO Zero library is included as a default component in the Raspberry OS. However, it is worth noting that this library is not limited to the Raspberry OS and may also be utilised on other platforms such as Mac OS or various Linux operating systems ([GPIO Zero](#RefGPIO)).

**CHAPTER 4: DESIGN SYSTEM AND METHODOLOGY**

**4.1. Hardware:**

Embedded computers are a diverse range of entities in contemporary society, serving many functions like the regulation of robotic systems, facilitation of the Internet of Things (IoT), and support of the automobile industry. Additionally, these products are created by several producers, each offering various pricing tiers. Two well-recognised and influential single-board computers in the field of computing are the Raspberry Pi, developed by the Raspberry Foundation, and the Jetson Nano, created by NVIDIA. The NVIDIA Jetson Nano and the Raspberry Pi are well-recognised single-board computers, both with distinct areas of emphasis and notable advantages. The selection between the two options is contingent upon the specific project needs, objectives, and financial considerations. Despite their diminutive size and use of the Linux operating system, these machines possess distinct benefits and downsides.

The Jetson Nano is a product produced by NVIDIA Corporation, a prominent business of this company known for its expertise in the production of graphics processors. NVIDIA Corporation is headquartered in the United States of America. The Jetson Nano is equipped with a high-performance Graphics Processing Unit (GPU), rendering it highly suitable for applications that need hardware-accelerated artificial intelligence and machine learning workloads. The tasks encompassed in this domain consist of object detection, picture recognition, and deep learning, with a particular emphasis on the contributions made by Nvidia. In addition, the Jetson Nano is equipped with specialised camera connections that enable the usage of from 4 to 6 high-resolution cameras. This feature distinguishes it from the Raspberry Pi, which only supports a single camera. Consequently, the Jetson Nano is particularly well-suited for computer vision tasks that need superior imaging capabilities. Nevertheless, the Jetson Nano does have several drawbacks, like its relatively greater energy consumption compared to other boards. This may be attributed to its increased number of GPU cores, which therefore contributes to its higher price. Furthermore, the Jetson Nano has just been produced, the result is its community has not been wide.

In contrast, the Raspberry Pi platform benefits from a substantial and engaged community, which has led to an abundance of resources, tutorials, and support that are driven by the community. Due to its versatility, the Raspberry Pi finds utility in a wide range of endeavours, including educational initiatives, hobbyist hobbies, and practical applications. In addition, it is worth noting that the Raspberry Pi utilises a singular graphics processing unit (GPU), resulting in less energy consumption compared to the Jetson Nano. Moreover, this characteristic also contributes to a cheaper overall cost of the Raspberry Pi. However, the utilisation of Raspberry Pi is accompanied by several drawbacks. The computational capabilities of the Raspberry Pi are insufficient for executing advanced artificial intelligence programs or handling the computational demands associated with the simultaneous processing of large-scale picture data from several cameras.

In summary, the Jetson Nano might be considered a suitable option for projects that need efficient processing of a large volume of high-resolution images, even if it entails a substantial financial investment. Alternatively, the Raspberry Pi might be a suitable option for applications that do not require high levels of performance and where cost is a significant factor to consider. The utilisation of Raspberry Pi is deemed appropriate for the implementation of this project.

4.1.1. Embedded computer Raspberry Pi:

The Raspberry Pi was the subject of research conducted by Eben Upton and his colleagues at Cambridge in 2006. Following a six-year development period, the inaugural commercial iteration, known as the Raspberry Pi 1 model B, was introduced in 2012. Over the course of a decade, four successive iterations, encompassing both Model A and Model B, have been developed. Currently, the most recent iteration of the Raspberry computer is the Raspberry Pi 4 model B ([Raspberry](#RefRa)). This particular computer type has several advantages, including reduced energy usage, minimal noise output, compact physical dimensions, and cost-effectiveness. Furthermore, the system is operated using open-source operating systems that are built upon the Linux kernel. This project utilises the Raspberry Pi 4 Model B with 8GB of RAM, running the Raspberry OS with the desktop (Legacy) Debian 10 (Buster) operating system.

According to Raspberry manufacturer, the main specifications of Raspberry Pi 4 include ([Raspberry](#RefRa)):

|  |  |
| --- | --- |
| Processor: | Broadcom BCM2711, quad-core Cortex-A72 (ARM v8) 64-bit SoC 1.5GHz |
| Ram: | 8GB LPDDR4 3200Mhz |
| Connectivity: | 2.4 GHz and 5.0 GHz IEEE 802.11b/g/n/ac wireless  LAN, Bluetooth 5.0, BLE  Gigabit Ethernet  2 × USB 3.0 ports  2 × USB 2.0 ports. |
| GPIO: | Standard 40-pin GPIO header |
| Video & sound: | 2 × micro-HDMI ports (up to 4Kp60 supported)  2-lane MIPI DSI display port  2-lane MIPI CSI camera port  4-pole stereo audio and composite video port |
| Input power: | 5V DC via USB-C connector (minimum 3A)  5V DC via GPIO header (minimum 3A)  Power over Ethernet (PoE)–enabled.  (Requires separate PoE HAT) |

A green circuit board with many small objects

Description automatically generated

Figure 12: Raspberry Pi 4 model B board ([Raspberry](#RefRa)).

Raspberry Pi is a compact computer. This means that it can be operated like a regular computer using the operating system core Linux (Figure X). It can be used for text editing, programming, and web browsing, even more, it can be served as a server. However, the great difference from the other computers is Raspberry Pi has 40 GPIO pins to send/receive signals ([Raspberry](#RefRa)). These pins can transmit/Receive signals via connecting standards such as UART, and I2C or supplying high/low (3.3V/0V) voltage.

A screenshot of a computer

Description automatically generated

Figure 13: Screenshot of Raspberry Pi OS (Debian 10 Buster)

A diagram of a circuit board

Description automatically generated

Figure 14: GPIOs Diagram of Raspberry Pi 4 ([Raspberry](#RefRa)).

4.1.2. Raspberry Pi Camera:

Raspberry Pi Cameras are the productions of Raspberry Pi ([Camera](#RefCa)), and they consist of vast different versions corresponding to different purposes such as fast motion photography, and infrared photography. In this project, the Raspberry Pi camera V2 has been used, and attached to Raspberry Pi 4 via a ribbon cable.

- Sensor: SONY IMX219

- Resolution: 8 MP.

A computer chip with a camera on it

Description automatically generated

Figure 15: Raspberry Pi Camera ([Camera](#RefCa))

4.1.3. Motor Driver L293N:

The Motor Driver L298Ns are popular H-Bridge integrated circuits to control two independent motors in bidirectional motors. The circuit has logic input pins to set the operating mode of the motors such as direct rotations and stop/run mode ([L298N](#RefL298)).

A red circuit board with white text

Description automatically generated

Figure 16: Motor Driver L298N Circuit ([L298N](#RefL298))

4.1.4. Servo SG90:

This project uses two tiny servos signed as SG90. The LxBxH dimension corresponds to 32x12x32 mm, and its weight is 15 grams, however, the torque comes up to 2.5 kg-cm. The power that they use is 5V and is controlled by the pulse chain [(Servo)](#RefSer). Nonetheless, they were adjusted from pulse control to direct control by reversing positive and negative poles to be appropriate for this project.

A diagram of a machine

Description automatically generated

Figure 17: Servo SG90 Dimension and Specification. [(Servo)](#RefSer)

**4.2. Design system:**

4.2.1. design description:

This robotic system has been engineered to enable control using manual hand gestures. The system comprises essential interconnected elements, including a camera, a Raspberry Pi 4 main board, a crawler chassis, a power source, two motor drivers, and two servos (see Figure 16). The robotic system is comprised of two distinct components, specifically the Input Signal Block and the Operating Block. The initial module, referred to as the Input Signal Block, encompasses the camera, Motor Driver 1, and two servos. The subsequent module, known as the Operating Block, comprises Motor Driver 2 and the Crawler Chassis. The operational process starts by using the Camera to capture a video stream of hand movements, which is subsequently sent to a Raspberry Pi 4 through a ribbon cable connection. Subsequently, the main board will undertake the task of analysing the frames inside the video in order to create two distinct categories of commands. The initial category of commands is transmitted to Motor Driver 1 of the Input Signal Block with the purpose of regulating the camera's focusing of hand movements. This helps the Camera stay focused on the hand even if the robot changes direction. The subsequent category of instructions will be transmitted to the Motor Driver 2 of the Operation Block in order to govern the locomotion of the Crawler Chassis, including actions such as forward movement, rightward rotation, and leftward rotation (see Figure 17).

**A diagram of a computer

Description automatically generated**

Figure 18: Blocks Diagram.

A diagram of a computer

Description automatically generated

Figure 19: Flow Process Diagram.

4.2.2. Control Methodology:

The robot has the capability to conduct four distinct acts, including turning left, turning right, moving forward, and coming to a halt. The previously mentioned tasks are regulated by four matching motions executed by the index finger. This indicates that the robot's leftward movement is activated when the controlling person tilts their index finger towards the left hand, while the robot's rightward movement is initiated when the index finger turns towards the right hand. Furthermore, the robot will go in a forward direction when the index finger is pointing upwards, and it will come to a halt when the hand is clenched into a fist, as seen in Figure 19.

A collage of hands pointing

Description automatically generated

Figure 20: Four Controlling Gestures.

**A diagram of a triangle

Description automatically generated**

Figure 21: Dividing Control Angles.

Based on the information shown in Figure 19, accurately identifying the proper direction by pointing the finger upwards in the context of the 'Go ahead' case creates a considerable challenge. Moreover, it is necessary to provide precise definitions for the inclination angles in the cases labelled as 'Turn-Left' and 'Turn-Right' in order to correctly distinguish between the actions of 'Turn-Left', 'Ahead', and 'Turn-Right'. This problem may be resolved by partitioning a 180-degree angle into three congruent angles, with each angle measuring 60 degrees (see Figure 19). The angle ABD indicates the direction to turn left, the angle DBE shows the direction to proceed straight ahead, and the angle EBC represents the direction to turn right.

The paper effectively presents a comprehensive overview of the hardware functionalities and fundamental design principles involved in the creation of a hand motion-controlled robot. The subsequent phase involves the deployment of the robot and the subsequent assessment of the obtained results.

**CHAPTER 5: DEPLOYMENT AND RESULT**

**5.1. Process of deployment:**

5.1.1. Wiring Diagram:

The motor drivers are responsible of controlling the rotation of the motors on the Crawler Chassis and the servos that regulate the camera direction, as seen in Figure 16's Block Diagram. The motor drivers, such as L298N and DRV8833, are activated by a voltage range of 2.3 V to 5 V. Consequently, it is necessary to establish a connection with the Raspberry Pi 4's pins that are initially set to a low-level voltage. This implies that the pins associated with Motor Drivers exhibit a low-level state subsequent to the initiation of Raspberry Pi 4. The designated pins, namely GPIO 17, GPIO 27, GPIO 22, and GPIO 10, are interconnected with Motor Driver 1 (DRV8833) in order to regulate the direction of the camera. The pins denoted as GPIO 23, GPIO 24, GPIO 25, and GPIO 9 are interconnected with Motor Driver 2 (L298N) in order to regulate the movement of the Crawler (see Figure 22, 23, 24). The pins will be designated and configured with a state of either "LOW" or "HIGH" by a program written in the Python programming language (shown in Appendix).

A diagram of a computer circuit

Description automatically generated

Figure 22: Wiring Diagram.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Connection | | Camera movement and states of pins | | | | | |
| Raspberry Pi | DRV8833 | Stop | Up | Down | Left Rot. | Right Rot. |
| GPIO 17 | IN1 | LOW | LOW | LOW | High | LOW |
| GPIO 27 | IN2 | LOW | LOW | LOW | LOW | High |
| GPIO 22 | IN3 | LOW | LOW | High | LOW | LOW |
| GPIO 10 | IN4 | LOW | High | LOW | LOW | LOW |

Figure 23: Wire Connection and States Table of Camera Control.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Connection | | Crawler movement and states of pins | | | |
| Raspberry Pi | L298N | Stop | Ahead | Turn-Left | Turn-Right |
| GPIO 23 | IN1 | LOW | HIGH | LOW | HIGH |
| GPIO 24 | IN2 | LOW | LOW | LOW | LOW |
| GPIO 25 | IN3 | LOW | HIGH | HIGH | LOW |
| GPIO 9 | IN4 | LOW | LOW | LOW | LOW |

Figure 24: Wire Connection and States Table of Crawler Control.

5.1.2. Inclined angles and focusing process:

As previously indicated in the design part, the locomotion of the robot was governed by the inclination angle of the index finger. Additionally, it is imperative for the camera to maintain focus on the index finger during the robot's motion. Hence, it is imperative to take into account the computation of inclined angle circumstances. Additionally, careful consideration must be given to determining the camera's position for the purpose of calibrating its direction.

The first problem is resolved by the determination of the angular inclination of the index finger, followed by a subsequent comparison with a predetermined angle of 60°. In the above scenario, the robot will proceed forward if the angle exceeds 60°; otherwise, it will execute a left or right turn. According to the data presented in Figure 23, the robot will initiate a turn when the angle α is less than 60 degrees. This may be expressed mathematically as tan(a) < tan(60o). Consequently, the robot will execute a right turn when the following requirements are met: xM < xN, yM > yN, and tan(a) < tan(60o), vice versa, it will perform a left turn when the following requirements are met: xM > xN, yM > yN, and tan(a) < tan(60o) (refer to Figure 25).

A diagram of a triangle

Description automatically generated

Figure 25: Illustration of Index Finger Inclined Angle.

|  |  |
| --- | --- |
| Motions | Satisfying conditions |
| Stop | yN > yM |
| Ahead | yN < yM and tan(a) > tan (60o) |
| Turn-Left | yN < yM, xN < xM, and tan(a) < tan (60o) |
| Turn-Right | yN < yM, xN > xM, and tan(a) < tan (60o) |

Figure 26: Conditions of Movements Table.

The next issue is to the camera's consistent focus on the hand during the robot's movement. The provided approach involves the establishment of a spatial area including the specific physical points corresponding to the wrist. This implies that should the wrist coordinates exceed the boundaries of the specified region in any direction, the camera will thereafter be manipulated in that particular direction. The region in requirement has dimensions of 150 pixels in height and 200 pixels in width. Additionally, it is positioned below the middle of the frame in the vertical direction and at the centre of the frame in the horizontal direction (see to Figure 26).

A blue rectangle with black text

Description automatically generated

Figure 27: Region of Wrist Coordinates on Image Frame.

**5.2. Result:**

After the installation and programming procedures were finished, the robot was subjected to testing, which produced the expected results. The integration of many components, including as the power system, Raspberry Pi 4, camera module, and motor drivers, has been successfully accomplished within the crawler mechanism. As a result, the robot is capable of independent operation.

Initially, the camera of the robot successfully maintained focus on the hand throughout its motion, which was being controlled by the hand directly. This functionality facilitates the camera's ability to precisely track the hand's gestures when it enters the camera's field of view. Additionally, the camera has the capability to capture the hand and render the palm of the hand within the frame of the streaming video. The installation of software on the Raspberry Pi4 enables the generation of commands for controlling the movement of the crawler (see Figure 28).

Furthermore, the crawler has successfully accepted orders from the Raspberry Pi4 and possesses the potential to be remotely operated. It can be guided through the use of gesture-based controls, including left and right turns, forward movement, and the ability to cease its operation.

The tasks performed by the robot demonstrate a high level of effectiveness in meeting the required requirements, which encompass both autonomous execution and remote-control using hand gestures provided by a camera.

A collage of hands pointing at a ceiling

Description automatically generated

Figure 28: Commands Generated by Hand Gestures.

**CHAPTER 6: CONCLUSION AND FUTURE WORK**

In conclusion, the research has helped strengthen the knowledge of the advance computer science course and created a robot which is the initial desire The robot may be manipulated using manual gestures to execute various orders, including forward movement, left turns, right turns, and stopping in real time. Moreover, this article has overcome the previous studies. Initially, it is not required for the operators to position themselves directly in front of a camera that is linked to a computer or utilise a cumbersome glove equipped with sensors and a battery in order to manipulate the robots. Furthermore, the integration of all components into the chassis of this robot enables it to operate autonomously with flawless execution. Nevertheless, it is important to acknowledge that this paper, like any other, is not without its imperfections, despite its inherent benefits. First and foremost, it is important to note that the camera utilised in this robot is of the fixed lens variety. Consequently, the robot's control capabilities are limited to a small range. When the distance between the controller and the robot increases, the robot's ability to identify hand signals is diminished. Subsequently, the camera, which is included in the robot, may encounter obstruction from an impediment, leading to potential difficulties in its functionality. As a result, this limitation prevents the robot from being deployed in environments with numerous obstacles. Nevertheless, the findings presented in this work will serve as a foundation for future investigations, particularly in the realm of using optical zoom systems or thermal imaging sensors. Moreover, the subsequent research has the potential to enhance the robot's control functionalities, enabling the operator to utilise both hands for controlling the robot. This would therefore facilitate the execution of more complicated tasks in real-time. Furthermore, this project establishes a foundation for future research that involves the direct implementation of artificial intelligence (AI) on robotic platforms.

**APPENDIX**

**Code of the main program:**

# import the necessary libraries

**from** picamera.array **import** PiRGBArray

**from** picamera **import** PiCamera

**import** cv2

**import** numpy **as** np

**import** mediapipe **as** mp

**import** camcontrol

# Calling function Picamera

camera = PiCamera()

# Setting resolution of camera

w = 1280

h = 720

camera.resolution = (w, h)

# Setting framerate

camera.framerate = 30

# Change frame to array

rawCapture = PiRGBArray(camera, size=(w, h))

# Calling function drawing utils of Mediapipe.slutions

mp\_drawing = mp.solutions.drawing\_utils

# Calling function hand of Mediapipe.slutions

mp\_hands = mp.solutions.hands

# set font

font = cv2.FONT\_HERSHEY\_PLAIN

# Calculating tan(60)

tan60 = np.tan(np.pi \* 60 / 180)

# create an list variable named LmList

lmList = []

# endless loop for streaming video

**with** mp\_hands.Hands(min\_detection\_confidence=0.8, min\_tracking\_confidence=0.5) **as** hands:

# capture frames from the camera

**for** frame **in** camera.capture\_continuous(rawCapture, format="bgr", use\_video\_port=True):

# grab the raw NumPy array representing the image

image = frame.array

# BRG to RGB

image = cv2.cvtColor(image, cv2.COLOR\_BGR2RGB)

# Detections

results = hands.process(image)

# RGB 2 BGR

image = cv2.cvtColor(image, cv2.COLOR\_RGB2BGR)

# deleting all components of lmList

lmList.clear()

# Rendering results

**if** results.multi\_hand\_landmarks:

# Drawing hand palm

**for** num, hand **in** enumerate(results.multi\_hand\_landmarks):

mp\_drawing.draw\_landmarks(image, hand, mp\_hands.HAND\_CONNECTIONS,

mp\_drawing.DrawingSpec(color=(0, 0, 0), thickness=2, circle\_radius=4),

mp\_drawing.DrawingSpec(color=(255, 255, 0), thickness=2, circle\_radius=2),

)

myHand = results.multi\_hand\_landmarks[0]

# getting coordinates from fucntion landmark

**for** id, lm **in** enumerate(myHand.landmark):

# calculating coordinates of key points of the hand

cx, cy = int(lm.x \* w), int(lm.y \* h)

# add id, cx, cy

lmList.append([id, cx, cy])

**if** len(lmList) != 0:

a = lmList[8][2] - lmList[5][2]

**if** a >= 0:

cv2.putText(image, 'Stop!', (150, 50), font, 2, (0, 0, 255), 2)

camcontrol.crawlerStop()

**else:**

a = np.abs(lmList[8][2] - lmList[5][2])

b = np.abs(lmList[8][1] - lmList[5][1])

# if index finger points straight up

**if** b == 0:

cv2.putText(image, 'Go ahead!', (50, 50), font, 2, (0, 0, 255), 2)

camcontrol.ahead()

**else:**

# if the index finger in the TURN-LEFT region

if (lmList[8][1] > lmList[5][1]) & (tan60 > a/b):

cv2.putText(image, 'Turn left', (50, 50), font, 2, (0, 0, 255), 2)

camcontrol.turnLeft()

# if the index finger in the TURN-RIGHT region

**elif** (lmList[8][1] < lmList[5][1]) & (tan60 > a/b):

cv2.putText(image, 'Turn right!', (50, 50), font, 2, (0, 0, 255), 2)

camcontrol.turnRight()

# if the index finger in the AHEAD region

**else:**

cv2.putText(image, 'Go ahead!', (50, 50), font, 2, (0, 0, 255), 2)

camcontrol.ahead()

# Control directions of camera

horizontal\_Position = lmList[0][1] # get horizontal position of wrist.

vertical\_Position = lmList[0][2] # get vertical position of wrist.

# region of wrist l x h 200x150

**if** horizontal\_Position < (w/2 - 100):

camcontrol.cameraRight()

**if** horizontal\_Position > (w/2 + 100):

camcontrol.cameraLeft()

**if** vertical\_Position < (h/2 + 100):

camcontrol.cameraUp()

**if** vertical\_Position > (h/2 + 250):

camcontrol.cameraDown()

# show the video window

cv2.imshow("Hand gestures", image)

# clear the stream in preparation for the next frame

rawCapture.truncate(0)

# if the `q` key was hit, break from the loop

if cv2.waitKey(1) & 0xFF == ord("q"):

break

#close the window

cv2.destroyAllWindows()

**Class of control:**

"""

This class obtains all methods that control the movements of the camera such as UP, DOWN, TURN LEFT, RIGHT and methods that control Crawler turn LEFT, RIGHT, AHEAD

"""

import RPi.GPIO as GPIO

import time

# setting up output mode using BCM

GPIO.setmode(GPIO.BCM)

GPIO.setwarnings(False)

# declare varialbes

c1 = 17 # GPIO 17

c2 = 27 # GPIO 27

c3 = 22 # GPIO 22

c4 = 10 # GPIO 10

m1 = 23 # GPIO 23

m2 = 24 # GPIO 24

m3 = 25 # GPIO 25

m4 = 9 # GPIO 9

# declare pinouts for controlling the Camera

GPIO.setup(c1, GPIO.OUT)

GPIO.setup(c2, GPIO.OUT)

GPIO.setup(c3, GPIO.OUT)

GPIO.setup(c4, GPIO.OUT)

# declare pin inputs for controlling Crawler

GPIO.setup(m1, GPIO.OUT)

GPIO.setup(m2, GPIO.OUT)

GPIO.setup(m3, GPIO.OUT)

GPIO.setup(m4, GPIO.OUT)

# method stopping the movement of the camera by setting output pins GPIO 17, 27, 22, 10 at LOW-level

**def** cameraStop():

GPIO.output(c1, GPIO.LOW)

GPIO.output(c2, GPIO.LOW)

GPIO.output(c3, GPIO.LOW)

GPIO.output(c4, GPIO.LOW)

# method stopping the movement of Crawler by setting output pins GPIO 23, 24, 25, 9 at LOW-level

**def** crawlerStop():

GPIO.output(m1, GPIO.LOW)

GPIO.output(m2, GPIO.LOW)

GPIO.output(m3, GPIO.LOW)

GPIO.output(m4, GPIO.LOW)

# method controlling camera moving UP by setting output pins GPIO 10 at a HIGH level and GPIO 22 at LOW-level

**def** cameraUp():

GPIO.output(c3, GPIO.LOW)

GPIO.output(c4, GPIO.HIGH)

time.sleep(0.01) # time for movement is 1 millisecond

cameraStop()

# method controlling camera moving DOWN by setting output pins GPIO 22 at HIGH level and GPIO 10 at LOW-level

**def** cameraDown():

GPIO.output(c3, GPIO.HIGH)

GPIO.output(c4, GPIO.LOW)

time.sleep(0.01) # time for movement is 1 millisecond

cameraStop()

# method controlling camera moving RIGHT by setting output pins GPIO 27 at HIGH level and GPIO 17 at LOW-level

**def** cameraRight():

GPIO.output(c1, GPIO.LOW)

GPIO.output(c2, GPIO.HIGH)

time.sleep(0.01) # time for movement is 1 millisecond

cameraStop()

# method controlling camera moving LEFT by setting output pins GPIO 27 at HIGH level and GPIO 17 at LOW-level

**def** cameraLeft():

GPIO.output(c1, GPIO.HIGH)

GPIO.output(c2, GPIO.LOW)

time.sleep(0.01) # time for movement is 1 millisecond

cameraStop()

# method controlling crawler AHEAD by setting output pins GPIO 23, GPIO 25 at HIGH level and GPIO 24, GPIO 9 at LOW-level

**def** ahead():

GPIO.output(m1, GPIO.HIGH)

GPIO.output(m2, GPIO.LOW)

GPIO.output(m3, GPIO.HIGH)

GPIO.output(m4, GPIO.LOW)

# method controlling crawler turnRIGHT by setting output pins GPIO 23 at HIGH level and GPIO 24, GPIO 25, and GPIO 9 at LOW-level

**def** turnRight():

GPIO.output(m1, GPIO.HIGH)

GPIO.output(m2, GPIO.LOW)

GPIO.output(m3, GPIO.LOW)

GPIO.output(m4, GPIO.LOW)

# method controlling crawler turnLEFT by setting output pins GPIO 25 at HIGH level and GPIO 23, GPIO 24, and GPIO 9 at LOW-level

**def** turnLeft():

GPIO.output(m1, GPIO.LOW)

GPIO.output(m2, GPIO.LOW)

GPIO.output(m3, GPIO.HIGH)

GPIO.output(m4, GPIO.LOW)

**BIBLIOGRAPHY**

[1] Sharkawy, A.N., 2021. Human-robot interaction: Applications. *arXiv preprint arXiv:2102.00928*. Available through website< [[2102.00928] Human-Robot Interaction: Applications (arxiv.org)](https://arxiv.org/abs/2102.00928)> [Accessed 26th July 2023]

[2] Irati, Z., Risto K., Alejandro H., Iñigo M., Lander U., Asier B., Víctor M., 2017. *Dissecting Robotics - historical overview and future perspectives*. Available through: ResearchGate website< <https://www.researchgate.net/publication/316538824_Dissecting_Robotics_-_historical_overview_and_future_perspectives>> [Accessed 15th June 2023].

[3] Kin-Huat L., 2007. *Industrial Robotics: Programming, Simulation, and Applications.* Mammendof, Germany.

[4] K. N. Lavanya, D. R. Shree, B. R. Nischitha, T. Asha and C. Gururaj, 2017. Gesture-controlled robot. 2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT), Mysuru, India, 2017, pp. 465-469, doi: 10.1109/ICEECCOT.2017.8284549. [online] Available at: < <https://ieeexplore.ieee.org/abstract/document/8284549/authors#authors> > [accessed 10th July 2023]

[5] Islam B., Tuton C. M., 2014. *Gyro-accelerometer-based control of a robotic Arm using an AVR microcontroller.* [2014 9th International Forum on Strategic Technology (IFOST)](https://ieeexplore.ieee.org/xpl/conhome/6975313/proceeding). Published by IEEE on [e-journal] <https://doi.org/10.1109/IFOST.2014.6991151>. [accessed 10th July 2023]

[6] Pranay I., Sanjana T., Swati D., 2019. *Hand Gesture Controlled* *Robot.* [2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT)](https://ieeexplore.ieee.org/xpl/conhome/8274997/proceeding). Published by IEEE on 14 May 2020. [e-journal] [https://doi.org/10.1109/ICET-SIP-1946815.2019.9092032](https://doi.org/10.1109/ICETET-SIP-1946815.2019.9092032). [accessed 20th July 2023].

[7] Chanhan G. and Chandhari P., *Gestures-based wireless robotic control using image processing*, *2015 5th Nirma University International Conference on Engineering (NUiCONE)*, Ahmedabad, India, 2015, pp. 1-7, doi: <https://doi.org/10.1109/NUICONE.2015.7449642>.

[8] K. N. Lavanya, D. R. Shree, B. R. Nischitha, T. Asha and C. Gururaj, *Gesture controlled robot*, 2017 International Conference on Electrical, Electronics, Communication, Computer, and Optimization Techniques (ICEECCOT), Mysuru, India, 2017, pp. 465-469, doi: <https://doi.org/10.1109/ICEECCOT.2017.8284549>.

[9] H. Zhao, J. Hu, Y. Zhang, and H. Cheng, *Hand gesture-based control strategy for mobile robots,* 2017 29th Chinese Control And Decision Conference (CCDC), Chongqing, China, 2017, pp. 5868-5872,doi: <https://doi.org/10.1109/CCDC.2017.7978217>.

[10] Gerard O’Regan, 2016. *Introduction to the History of Computing: A Computing History Primer.* Published by: Springer Nature, Switzerland.

[11] Milan S., Vaclav H., and Roger B., 2015. *Image Processing, Analysis, and Machine Vision.* Published by: Cengage Learning, Stamford, USA.

[12] Marc P. D., A. Aldo F., and Cheng S. O., 2020. *Mathematics For Machine Learning*. Published by Cambridge University Press.

[13] Stack Overflow. *Stack Overflow Trends*. [online] Available at <https://insights.stackoverflow.com/trends?tags=c%23%2Cjava%2Cjavascript%2Cpython%2Cc%2B%2B>. [Accessed 18 August 2023]

[14] Mark Lutz, 2009. *Learning Python*. Fourth Edition. Published by O’Reilly Media, USA.

[15] Google Mediapipe. *Mediapipe Solution*. [online] Available at <https://developers.google.com/mediapipe/solutions/vision/hand_landmarker#get_started>. [Access 15th June 2023].

[16] Ivan Idris, 2014. *Learning NumPy Array*. Published by Packt Publishing Ltd., Birmingham, UK.

[17] NumPy Organization. *NumPy Array*. [online] Available at <https://numpy.org/doc/stable/user/whatisnumpy.html>. [Accessed 18th June 2023].

[18] Joe M., Joseph H., 2015. *Learning OpenCV 3 Computer Vision with Python*. Second Edition. Published by Packt Publishing Ltd., Birmingham, UK.

[19] OpenCV Organization. [online] Available at <https://opencv.org/about/>. [Accessed 20th July 2023]

[20] NVIDIA Corporation. Jetson modules. [online] Available at <https://developer.nvidia.com/embedded/jetson-modules>. [Access 16 August 2023]

[21] Website of Raspberry Ltd. *Documentation category of Raspberry Pi*. [online] Available at <https://www.raspberrypi.com/documentation/computers/raspberry-pi.html>. [accessed 16 August 2023]

[22] Website of Raspberry Ltd. *Pi Camera module 2*. [online] Available at <https://www.raspberrypi.com/documentation/computers/raspberry-pi.html>. [accessed 16 August 2023]

[23] Motor Driver L298N H-Bridge Integrated Circuit. [online] Available at <http://www.handsontec.com/dataspecs/L298N%20Motor%20Driver.pdf>. [Accessed 17 August 2023]

[24] Servo SG90. [online] Available at <http://www.ee.ic.ac.uk/pcheung/teaching/DE1_EE/stores/sg90_datasheet.pdf>. [Accessed 17 August 2023]

[25] Motor Driver DRV8833 H-Bridge. [online] Available at <https://users.ece.utexas.edu/%7Evalvano/Datasheets/drv8833.pdf>. [Accessed 21st August 2023]

[26] GPIO Zero. [oline] Available at <https://gpiozero.readthedocs.io/en/stable/>. [accessed 23rd August 2023]