**Real-time Image Processing for Autonomous Guidance of Maze Following Robot**

**Introduction**

**Project Outline:**

The title of the project signifies that we aim to design an autonomous robot that is nothing but an intelligent machine that knows how to perform specific tasks without having direct control by humans in real-time. Autonomous robots offer quite significant applications and their implementations in the fields like locomotion (flying, crawling, and wheeled robots), navigation, mapping, orientation & object detection.[2] Moreover, autonomous robots have also shown their usage in cleaning as autonomous vacuums or vacuum cleaners. [1] One of the major applications of autonomous robots can be seen as autonomous vehicles like self-driving cars whose ultimate goal is to cultivate a technology that could enhance road efficiency and lessen traffic accidents. [3]

In this project, the autonomous robot is being guided to move inside the maze by detecting walls that come in front of it and then making either a right or left turn accordingly. First, the robot takes the images of the maze in real-time from the camera mounted on it. There will be multiple images of the maze taken by the robot’s camera. For this purpose, Raspberry Pi is to be connected to the motor using the bus cable. By using python code, we can enable the robot’s camera and take pictures of the maze as per the requirements. The pictures of the maze taken by the robot would be generic digital images similar to the images taken by a normal camera. Then each image is gone through image processing which is a method of performing any particular operation on an image so that we could get an enhanced yet desirable image as per our requirements by extracting fruitful information from it. However, image processing is a tool that helps us fetch the information or data from the image that could help our robot to perform its operations autonomously. Therefore, the main objective of the robot is to follow the algorithm that is written for it to detect walls. The Wall detection algorithm combines multiple steps in which edge detection is one of the vital parts of it. Using image processing, the image of the maze is taken and converted into a grayscale image which can later be used for edge detection.

**Objectives:**

The main objectives that we aim to achieve in this project include image processing in real-time so that the robot could behave like an intelligent machine without straightforward human control and make the decisions to travel in the maze by avoiding any obstacle in the form of walls. The first and foremost aim to achieve in the project is to capture representative images of the maze for the robot. This is an initial step as for the robot to start following the maze, it must have the images stored in it to make the decision accordingly. The objective here is to first take the images from the robot’s camera and then do image processing with it. Once we are successful in taking the representative images from the robot’s camera it will be easier to implement an image processing tool in it. Ideally, the robot is supposed to take real-time images of the maze from its camera, and then image processing is done for every image taken by it. The main advantage of real-time image processing helps the robot navigate through the maze without getting into the phase of preprocessing. The task of preprocessing would have involved taking the images of the maze beforehand and then with image processing, the path would be formed for the robot to follow. However, in real-time image processing, the robot takes the images of the maze in real-time, does image processing, and follow the maze. The other main goal to achieve in this project would be an algorithm for the robot for wall detection. The whole concept of image processing in the project is linked with writing the wall detection algorithm. Previously, maze-solving robots have been designed like line following robots or robots that use maze-solving algorithms. In line following robots, a path/line is already defined in the maze for the robot, and sensors are mounted on it and with the help of the controller, it solves the maze. On the other hand, one of the most popular maze-solving algorithms is used called the wall-follower algorithm in which a robot focuses on either the right wall (Right-Hand Rule) or the left wall (Left-Hand Rule). [4] Wall-follower algorithm also requires sensors in the structure of the robot to detect the wall by receiving the information from the external environment. [4] However, in this project, we focus on writing an algorithm for the robot to detect features (walls) without the help of any controller or sensor. Our goal is to design a robot that takes the images and by using Raspberry Pi, real-time image processing is done and the robot would detect the wall ahead of it by the algorithm written for it rather than solving the whole maze. The algorithm includes an edge detection mechanism for the images of the maze, by doing so, this edge detection tool enhances the edges of the maze for the robot to recognize the wall ahead of it. Writing a wall detection algorithm could be a daunting task as it defines the precision of the robot to take left, right, forward and backward movements. Once an algorithm is successfully written for the robot to detect walls, it is required to define the logic for the robot’s movement. These logics are defined in the way to guide the robot to make a turn by first evaluating a wall and since it would be capturing the images in real-time as it moves inside the maze, it would take turns accordingly. Therefore, by taking real-time images, image processing, and following the wall detection algorithm, the robot would be able to follow the maze. The last objective would be the implementation of this entire coding on the robot.

**Motivation / Background:**

As previously mentioned, there have been numerous autonomous maze-solving / maze-following robots that gave the motivation for this project. Maze-solving robot, also known as a micro-mouse is based on the function to find the route in the maze itself as its coding is done in a way that allows it to interpret the path on its own without taking any help or assistance. This autonomous maze-solving robot is already serving mankind in the applications like pipe inspection, bomb disposal, material handling, carrying goods, and warehouse management. This kind of maze-solving robot is quite unlike the line-following robot in which the robot has to follow a predefined path. It works on the logic done in Arduino code and ultrasonic sensors are used to help navigate the robot in the maze [5]. Another maze solver robot has been implemented using Artificial intelligence with Arduino which operates on two basic principles i.e. the robot is responsible to locate its way out of the maze no matter where it is put inside the maze and as soon as a robot finds its path, it ought to enhance the solution by finding the shortest path from start to end. This maze-solving robot is designed to first find the walls of the maze and then follow them to move inside it by implementing either left or right hand on the wall algorithm at Arduino code with the help of multiple sensors. First, the algorithm allows the robot to follow the wall by the left/right-hand rule to complete the route in the maze, and next time by using artificial intelligence, the robot would be able to take the shortest path [6]. Then there comes an autonomous robot that uses image processing and a pathfinding algorithm to find the best route for maze solving. In this method, a top view image of the maze is taken, and then preprocessing is done to find the path from start to end point. The image processing in this method involves preprocessing of the image which gives a virtual line (well-defined path in the image) for the robot to follow. After that, an algorithm is designed for the robot to accomplish the aim of traversing [7].

So far, we have seen autonomous robots that have been built to solve the maze using sensors, maze solving algorithms, Arduino, or preprocessing of the image. One of the many reasons that became the motivation of this project was not just aiming to solve the maze using any pre-defined or virtual path, but it’s rather focused on feather/wall detection without the help of sensors. Furthermore, in this project, Raspberry Pi is being used for real-time image processing. The purpose of real-time image processing is quite different than preprocessing in which a robot is already defined with a path to escape through a maze. In this project, we are motivated to write a wall detection algorithm with the help of image processing. Not only is image processing responsible for providing real-time images of the robot’s camera but it is also involved in writing the logic for the robot to detect walls in front of it and move in a maze without hitting them or any edges.

**Theory:**

1. **Raspberry-Pi 3 Model B+:**

Raspberry Pi is nothing but a small-sized computer having a single board that runs Linux on it. It is quite a cheap and affordable device to help programmers operate electronic devices, physical computing, and the Internet of Things (IoT). [8] Raspberry Pi gets easily plugged into a desktop monitor or a normal TV and can be operated using a computer’s keyboard and mouse. Having the ability to interact with the outside world, Raspberry Pi is capable of doing everything one would expect from a normal computer [8]. It allows its users to do programming in Python to control a robot, helps students to run their hardware projects, is used as a gaming server, helps take pictures, makes videos, etc [9]. In this project, Raspberry Pi 3 Model B+ has been used which is a 3rd generation single-board computer with the following specifications:

* 1.4GHz 64-bit quad-core processor
* Wireless LAN
* Bluetooth 4.2
* 40-pin GPIO (General Purpose Input Output)
* Camera Serial Interface (CSI), Camera port for connecting a Raspberry Pi camera
* Display Serial Interface (DSI), Display port for connecting Raspberry Pi display screen
* HDMI port
* 4 USB 2.0 ports
* 5V / 2.5A DC power input [10]

**Processor:** This is the basic part of Raspberry Pi which is responsible for all the computations, decision making, data processing, and fetching the results. BCM2837B0 in this Raspberry Pi is a fast and power-efficient quad-core processor running with a clock speed of GHz.

**GPU:** It is a hardware component that is used to perform operations on images and videos that’s why it is called a Graphics Processing Unit. GPU with OpenGL (Open Graphics Library) version 2.0 and hardware-accelerated OpenVG API which is a 2-D Vector Graphics Application Programming Interface is used in this Raspberry Pi 3 B+.

**Storage:** Raspberry Pi 3 B+ offers a storage capacity of 1GB and RAM used in it is LPDDR2 SDRAM which is the low power double data rate synchronous dynamic random access memory designed to work on low power without risking performance.

**Wireless LAN & Bluetooth 4.2:** Raspberry Pi 3 B+ comes with built-in Bluetooth and Wi-Fi features, allowing its users to set up Wi-Fi wirelessly. The on-board connectivity options of Raspberry Pi 3 B+ provide with successful set up of wireless networks in terms of both Wi-Fi and hotspot without using any kind of USB dongles.

**HDMI & USB Ports:** Raspberry Pi 3 B+ consists of 4 USB ports so there would be no need to connect extra ports using the USB hub. Standard HDMI (Type A) connector is used in Raspberry Pi 3 B+ model. The standard (Type A Male to Type A Male) cable helps Ethernet, 3D, and 4k videos.

**Power Input:** Raspberry Pi 3 B+ requires a power supply of 5V/2.5A for the Pi to work and a minimum of 2A would be sufficient to maintain its stability. [11] &[12]

**40-pin GPIO (General Purpose Input Output):**

To interface the robot with the Raspberry Pi so that robot can be operated as per the coding done on the Raspberry Pi, it is necessary to connect the 40-pin GPIO bus cable. It is called GPIO (General Purpose Input Output) because these pins provide standard interfacing by connecting the Raspberry Pi to external input/output devices. The GPIO has power pins such as 5V pins, 3V pins, and 0V (ground pins) and plenty of input/output pins. The processor of the Raspberry Pi works on 3V and if a sensor that runs on 3.3V is used then this power pin becomes vital for it. However, the Raspberry Pi board needs 5V to get powered with also the 5V power pin can be used as a supply for the motors that need 5 volts to run. Then there are TX and RX are the transmission and receiving pins that are used for the communication. The number of input/output multifunctional pins used with I2C (Inter-Integrated Circuit) and SPI (Serial Peripheral Interface) communication protocols.

**Operating System:**

Raspberry Pi 3 B+ contains an operating system specially made for it and the Linux it has is Debian version 11 and the version of kernel it supports is 5.15. The operating system in this Raspberry Pi is Raspberry Pi OS lite and it doesn’t have GUI (Graphical User Interface) in lite version.

**SSH:**

1. **Motor:**

To control the robot's movements, DC motors are being used in its hardware. (remaining)

1. **Camera:**

To take the digital images of the maze, a Raspberry Pi camera is used which has a Raspberry Pi operating system, a derivative of Debian. The Raspberry camera board allows direct connection to the Raspberry Pi. It comes with a 15cm flat strip ribbon cable with a 15-pin MIPI CSI connector that goes into the CSI port on the Raspberry Pi, specifically made for interfacing with the cameras. The camera module has 5MP (Million Pixels) and state image resolution is 2592 x 1944 and gives an HD (High Definition) image i.e. 1080p.

1. **Image Processing:**

As previously discussed, image processing is a technique in which an image is taken from a digital camera or through scanning and then can be reconstructed, manipulated, and enhanced into a desirable form. In this project, its main task is to compress or analyze the taken image and collect the information from it so that it can be later used for writing an algorithm for wall detection. The first step in image processing was edge detection as it is the fundamental step in extracting information from the images. In edge detection, the regions in the image where there are differences in the intensities of the pixel are identified as edges by using the matrix method.

**Libraries**

When it comes to programming using Python, there are quite useful libraries available to perform which are specifically designed to perform certain tasks. Similarly, numerous libraries come in handy for image processing as well which could make a programmer’s life easier. One such example is the Open CV (Open source Computer Vision), this library is widely used on many different platforms to process, enhance or reconstruct an image. The project for the Open CV library was first initiated in early 1999 by Intel researchers for the advancement of CPU-intensive applications. The library was originally designed to contain functions that programmers could use to help them with real-time computer vision. Open CV comes with a feature of GPU acceleration for real-time operations that was initially started in 2011 whose purpose is to enhance the speed of the processing of an image, bluffing it as real-time processing. Many useful implementations can be achieved using the Open CV library, some of them are listed below:

* Detection of a face
* Detection of edges
* Adding filters
* Removing watermark from an image
* Converting images to greyscale or binary
* Detection of a certain color in an image
* Removing background noise in an image
* Converting a regular image into a cartoon

In this project, the Open CV library is used to carry out the processing of the captured images of the maze using a digital Raspberry Pi camera. This processing mainly includes finding the edges of a wall which would help later to guide the robot autonomously to avoid collision and do the movements accordingly.

**Edge detection**

[13] Edge detection is one of the primary tools of image processing in which the amount of data or pixels is reduced to process and conserves the structure of the image. Edge detection itself is an algorithm containing the following steps:

1. **Filtration:** Digital images taken by the camera need to be filtered out as they might contain various oddities such as noise that might reduce the performance of an edge detector. However, filtering needs to be done in such a way that an image doesn’t lose the strength of the edges. In the images of the maze, there had been quite a few shadows of the light which were filtered out while doing edge detection.
2. **Enhancement:** Before getting into the final step of edge detection, it is important to recognize the change in intensities of the pixel in the taken image. Therefore, the enhancement mode focuses on indicating how brighter or lighter the points are than the other in an image. Since the blocks of the maze were white-colored, there had been a change in intensity between the front block and the blocks on the sides of it. Therefore, enhancement was needed to emphasize the pixels in the image with a major change in local intensity.
3. **Detection:** To detect the edges, points with strong edges are to be considered. It may be the case that an image has a few points that are not qualified as edges for the desired application [13]. It does a thresholding operation by extracting the edge pixels, qualified to be reserved and the rest should be rejected as noise.

There are multiple functions available to carry out image processing for edge detection, each with its pros and cons. The three widely used edge detection functions in Open CV are as follows:

* Sobel
* Canny
* Laplacian

**Sobel Edge Detector:**

The Sobel edge detector is a gradient-based method to detect edges using first-order derivatives. It measures the gradient of an image which is a directional change in the image intensity to highlight the areas of maximum rate of change of image intensity values in the spatial domain that relates to edges. It determines the first derivative of the image in 2D by taking the derivative of both X and Y axes. Sobel edge detector feature uses a pair of 3x3 convolution kernels for the derivatives, one for Y (vertical) axis and the other for X (horizontal) axis. However, convolution is a method of multiplying two arrays with different sizes but the same dimensions to produce a resultant array having a similar dimension to the two arrays. Whereas, kernel is a matrix used in image convolution, usually a small matrix with different sizes and configuration of numbers. [14] & [15]. In gradient-based edge detection methods, when there is a peak or steep, it is declared as an edge pixel but the drawback of such techniques is that it can be difficult to determine what is a peak and what isn't that’s why it’s hard to discriminate between a peak (edge) and noise[16].

**Laplacian Edge Detector:**

Laplacian is a second derivative operator that is used to find edges in the image. Laplacian calculates second-order derivatives, which distinguishes it from other edge detector operators, and unlike Sobel, it uses a single kernel for the derivative. The other different feature of laplacian that sets it apart from the other edge detection operators is that the edges it highlights are random, either inward or outward with no particular direction. Laplacian comes with the idea of zero-crossing which represents a point of the edge location where a picture's intensity fluctuates quickly. However, the one disadvantage of using laplacian is that since it takes a second-order derivative, it becomes extremely sensitive to noise. [16] & [17]

After measuring the pros and cons of all three canny was decided as the best algorithm for carrying out edge detection in this project.

**Comparison of Edge detection methods:**

As discussed earlier, it is evident that edges have a higher pixel intensity than the other points in its surrounding. Generally, edge detection techniques are classified into two categories:

1. Gradient-based edge detectors
2. Laplacian edge detectors

Gradient-based operators take the first derivative of the image with respect to time ‘t’, and the first derivative is at a maximum at the center of the edge. Then the threshold value is set to compare with the gradient value, if the gradient value exceeds the threshold value, an edge is detected. However, if the first derivative is at a maximum then the second derivative would be zero. Therefore, the laplacian operator locates the edges by finding the zero crossings in the second derivative [18]. Examples of gradient-based edge detectors are Sobel, Roberts, Prewitt, and Canny, whereas Marr-Hildreth / Laplacian of Gaussian (LoG) and Zero Crossing (ZC) are laplacian-based operators. All three renowned edge detectors were examined prior to choosing canny for the project's edge detection, and it was determined that canny produced the best results despite time-consuming and complex computations. Given that the Sobel and laplacian are both quite sensitive to noise, the canny operator detects weak edges even in noise conditions, better in calculating error rate and improving signal to noise ratio. Canny is also able to detect continuous, thin edges as well as edges in the black areas that the other edge detection operators were unable to pick up on. [19] & [20] & [21]. Normally the gradient-based operators tend to detect thick edges but due to non-maximal suppression, canny is capable to detect thin and smooth edges in the image. Another big difference between commonly used Sobel and canny operators is that Sobel's final output is a blurred image whereas, canny gives the clarity of the image by producing an output with sharp edges. [22] (Radhika chandwakar)

**Canny Edge Detector:**

For edge detection, the canny operator was chosen as it is the most commonly used and seemed to be the most effective method for detecting walls of the maze among all available edge detection operators. Canny edge detector maximizes the probability of detecting the actual edges and minimizes the chances of highlighting the non-edgy points in the image thus reducing the error rate. Canny edge detection is a technique used to filter the edges in the image. It also ensures localization by detecting the edge closer to the actual edge. Lastly, it is responsible to generate a single edge detection response and avoid creating multiple responses to a single edge point. Moreover, it is an edge detection method consisting of multiple algorithms as follows:

1. **Smoothing & Noise Removal:** The initial stage in edge detection is to eliminate undesired noise from the original image; as a result, blurring is used to filter out the noise and provide a smooth output. Canny edge detector accomplishes this using a Gaussian filter.
2. **Finding Gradients of the Image**: The next step is to find the gradients of the image which is done by taking the first derivative with the help of the sobel kernel. This phase involves taking the first derivative for computing vertical and horizontal components as well as the magnitude and direction of the gradient. The direction of the gradient is perpendicular to the edges and it is rounded to a single one of the four angles demonstrating directions like horizontal, vertical, and two diagonals. (meaning)??
3. **Non-Maximal Suppression:** Once getting the magnitude and direction of the gradient, the next step is to get rid of the thick edges in the source image, and this process of thinning the edges can be done by non-maximal suppression. Moreover, the local maxima are located in the direction of the gradient and designated as edges, while the others are suppressed, lowering the number of false edges. [22] & [23]
4. **Double Thresholding:** This step is called double thresholding as two threshold values are taken in it to decide what actual edges are and what needs to be discarded. The two threshold values are called T1 = Max. Value and T2 = Min. Value respectively. The pixel intensity values of the grayscale are then compared with the threshold values. If the values of pixels are greater than T1 then the pixels are declared to be the strong/sure edges and if the values of gray scale are less than T2 then pixels are weak / non-edges. If the values of the pixels lie between T1 & T2 then the outcome depends on the neighboring pixels.
5. **Hysteresis Edge Tracking:**  Once it is decided what strong and weak edges are, strong/sure edges are considered in the final output image, and weak/uncertain edges that are not linked with the strong edges are rejected. However, weak edges that are connected to the strong edges are included in the final output image. Hence, it is very important to choose threshold values to get accurate results. [22] & [24]

**Anaconda:**

Anaconda is a software available to allow the programmers to work in different environments where it is easier to write codes in Python that’s why it is commonly known as Python distributor. It is a free software that comes with Python interpreter, various Python packages, and editors. After installing Anaconda, we gain access to environments called as Integrated Development Environments (IDEs) which are capable of giving comfort in creating codes for different projects. The useful functions offered by IDEs, such as the ability to develop, edit, and debug codes, view and examine data, store variables, and present results, are always in demand. Before Anaconda, Cpython served as the Python distribution tool for programmers. HoweverAnaconda is a relatively straightforward Graphical User Interface (GUI) that offers useful, well-sustained libraries and IDEs. Important libraries like Scipy, Numpy, and Matplotlib are all present in Anaconda, along with the open-source python programme called Biopython. With a single installation, Anaconda eliminates the need for several Python installations with various libraries, functions and IDEs. [25] & [26]

Spyder, jupyter and anaconda(more)