A Project Report

on

RASPBERRY PI – BASED READER FOR THE VISUALLY IMPAIRED

by

R. SAMUEL	**20F85A0402**
A. GUNASEKHAR	**19F81A0404**
V. GUNASEKHAR	**19F81A0405**
Sk. RESHMA	**19F81A0406**

Submitted to the Jawaharlal Nehru Technological University Anantapur, Ananthapuramu
in partial fulfillment of the requirement for the award of degree of

BACHELOR OF TECHNOLOGY
IN
ELECTRONICS AND COMMUNICATION ENGINEERING

Under the esteemed Guidance of
Mrs. M. Gnanapriya, M.E , M.I.S.T.E , M.I.S.S.E
Vice Principal & Dean, GKCE

DEPARTMENT OF ECE
GOKULA KRISHNA COLLEGE OF ENGINEERING
(Affiliated to JNTUA, Ananthapuramu and Approved by AICTE, New Delhi)
Sullurpeta, Tirupathi District, A.P. – 524121.

May - 2023

GOKULA KRISHNA COLLEGE OF ENGINEERING, SULLURPET

(Affiliated to JNTUA Ananthapuramu and Approved by AICTE New Delhi)

DEPARTMENT OF ECE

BONAFIDE CERTIFICATE

This is to certify that the project entitled "RASPBERRY PI – BASED READER FOR THE VISUALLY IMPAIRED" is a bonafide work done by

R. SAMUEL	20F85A0402
A. GUNASEKHAR	19F81A0404
V. GUNASEKHAR	19F81A0405
Sk. RESHMA	19F81A0406

In the partial fulfillment for the award of **BACHELOR OF TECHNOLOGY** in JAWAHARLAL NEHRU TECHNOLOGICAL UNIVERSITY ANANTAPUR, ANANTHAPURAMU.

This work hasn't been submitted for the award of any other degree.

Project Guide **Head of the Department**

Submitted for the university examination (viva-voce) held on _____.

External examiner

ACKNOWLEDGMENT

Motivation and inspiration provided by everybody has helped us in the successful completion of our project work. We would like to acknowledge several people who have helped us in this project work, failing which it would not have shaped up the way it has today.

We are grateful to convey our sincere thanks to **Dr. M. Suresh, M.Tech., Ph.D., Principal, GKCE** for being a source of inspiration.

We are gratified to thank **Dr. M. Chiranjeevi, Ph.D., Head of the Department of Electronics and Communication Engineering** for his valuable guidance and suggestions.

We would express our sincere thanks to our guide **Mrs. M. Gnanapriya, M.E, M.I.S.T.E., M.I.S.S.E., Dean and Vice Principal, GKCE** for her kind guidance for planning and implementation of the project.

We extend our sincere thanks to all the lab technicians of ECE Department for their continuous support and kind co-operation throughout the project duration.

ABSTRACT

According to the World Health Organization (WHO), At present at least 2.2 billion people around the world have a vision impairment, of whom at least 1 billion have a vision impairment[1]. A further 110 million have low vision and are at great risk of becoming blind. These people have a very hard time using their computing devices because seeing the information displayed is very crucial. The majority of published printed works do not include Braille or audio versions and digital versions are still a minority. The already existing systems such as screen readers and visual aid equipments are either expensive, can only convert texts in physical papers or require additional hardware / software to convert both physical and digital copies using the same device.

To make these digitally or physically printed works accessible to the visually impaired and blind people, Raspberry Pi-based Reader is used, which would convert physical or digital printed materials into digital audio. In order to accomplish this goal, an image processing library, an Optical Character Recognition (OCR) software and a Text-To-Speech (TTS) engine are used.

OpenCV is a open source computer vision and image processing library that is being used to help the OCR tasks. OCR enables the recognition of texts from image data and is used to convert text information in digital or physical photos to electronic text files for the TTS. The TTS engine enables a text in digital format to be synthesized into human voice and played through an audio system.

Keywords: Visually Impaired, Blind, Raspberry Pi, OCR, OpenCV, TTS, Reader

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1
INTRODUCTION

1.1 OVERVIEW OF THE PROJECT

According to the World Health Organization (WHO), At present at least 2.2 billion people around the world have a vision impairment, of whom at least 1 billion have a vision impairment[1]. The majority of published printed works do not include Braille or audio versions and digital versions are still a minority. So, Raspberry Pi-based Reader is being used to make these digitally or physically printed works accessible to the visually impaired and blind people. Raspberry Pi 3 Model B+ board is the heart of this project. The Raspberry Pi Camera is connected through CSI port acts as an eye for the blind and visually impaired. A simple push-button switch is used to make the device easily accessible to these people.

In the software side of things, OpenCV, a open source computer vision, machine learning and image processing library is being used to help the Optical Character Recognition tasks by performing image manipulations and de-noising the input images. Tesseract OCR enables the recognition of texts from image data and is used to convert text information in digital or physical photos to electronic text files for the Text-To-Speech engine. The eSpeakNG TTS engine enables a text in digital format to be synthesized into human voice and played through an audio system. All these software libraries are used from within Python program using the respective Python modules and wrappers.

When the push-button is pressed once, Pi Camera is activated and captures an image. The captured image is processed using OpenCV, text is extracted using Tesseract OCR and speech audio is given as output using eSpeakNG TTS engine. The device can also convert the texts in the active (currently used) window into audio. This is be done by pressing the push-button twice. After a screenshot is captured, the same procedure is followed to convert texts in image into audio.

1.2 LITERATURE SURVEY

- **Document Segmentation and Language Translation Using Tesseract-OCR[2]**

Sahil Thakare et. al, in 2018, used Python scripting, Python-tesseract library and Google translate to segment document and perform translation, which are one of the key areas in pattern recognition and natural language processing. Translation in terms of a web application that accepts image document as an input, where input document is a user define image file containing text in any language and does its exact translation in any supported languages using Google Translator. and various libraries are used to approach various challenges in segmentation and translation of a document.

- **Optical Character Recognition Using Tesseract[3]**

Jally Brahmani et. al, in 2022, used Tesseract OCR, Long-Short Term Memory (LSTM) neural network and Android to to allow automatic extraction of the information that a user wants from the paper document and using it wherever it is needed. Optical Character Recognition (OCR) is a process or technology in which text within a digital image is recognized. An Android app was made, which provides seamless experience (no advertisements and easy-to-use) and great accuracy. Tesseract is used to display the text to the user and uses a deep learning model to classify the letters and display them to the user. It adds a new neural network (LSTM) based OCR engine which is focused on line recognition but also still supports the legacy Tesseract OCR engine which works by recognizing character patterns.

- **Extract Text from Images in Python using OpenCV and EasyOCR[4]**

Himanshu Nath Tiwari in April 2023, used Open Source Computer Vision and EasyOCR to Extracting text from images has many applications, such as in optical character recognition (OCR), document digitization, and image indexing. In this paper, OpenCV and EasyOCR libraries were used to extract text from images in Python.

- **Smart Cap - Wearable Visual Guidance System for Blind[5]**

A. Nashajith et. al, in 2018, used TensorFlow API and eSpeak TTS in Raspberry Pi 3. The people who are having complete blindness or low vision faces many difficulties during their navigation. The main purpose of this paper was to develop a navigation aid for the blind and the visually impaired people. In this paper, a smart cap is implemented, which helps the blind and the visually impaired people to navigate freely by experiencing their surroundings. The scene around the person will be captured by using a NoIR camera and the objects in the scene will be detected. The earphones will give a voice output describing the detected objects. The architecture of the system includes the processor Raspberry Pi 3, NoIR camera, earphones and a power source. The processor collects the frames of the surroundings and convert it to voice output. The device uses TensorFlow API, open-source machine learning library developed by the Google Brain Team for the object detection and classification. TensorFlow helps in creating machine learning models capable of identifying and classifying multiple objects in a single image. Thus, details corresponding to various objects present within a single frame are obtained using TensorFlow API. A Text to Speech Synthesiser (TTS) software called eSpeak is used for converting the details of the detected object (in text format) to speech output. So the video captured by using the NoIR camera is finally converted to speech signals and thus narration of the scene describing various objects is done. Objects which come under 90 different classes like cell phone, vase, person, couch etc are detected.

1.3 EXISTING SYSTEMS

There are some already existing commercial readers for visually impaired some of which are recommended by the country. There are also some DIY solutions available but most of them don’t work now as they are not kept up-to-date with the latest software library changes. So lets see the two types commercial systems, that are mostly recommended by various governments.

- **ReadIt Air**[6]

ReadIt Air a compact, lightweight portable camera (shown in figure 1.1) that works with the ReadIt software and enables instant access to printed documents converting them to large print and text-to-speech output. It has been specifically designed for the requirements of students, business professionals and anyone on the move with a laptop PC. Its attachable positioning guide and custom designed positioning mat also allows users with no vision at all to perfectly place and capture their documents. With digital capture and OCR of A4 documents, the spoken and reformatted large print text can be read over twice as fast as traditionally enlarged text from standard video magnifiers. It also allows you to read for longer periods with much less eye strain. Users get the benefit of both automatic scrolling and speech output of their document.

Figure 1.1: ReadIt Air

It can capture 30 pages per minute and store entire text books or long documents in minutes. Once they're captured. It even remembers your position in each document when you re-open it and uses the latest camera and recognition technology to ensure it is as accurate as possible, even on complicated documents newspaper layouts or tins of food. It occupies 65mm x 62mm of table area. The ReadIt Air is recommended by Australian Disability Clearinghouse on Education and Training (ADCET)[7] for blind and visually impaired people. It is easily usable and has a intuitive interface but it still requires some amount of user interaction with computer. It can convert physical papers into text (which is read back by a natural-sounding TTS). It costs around A$2,880 (₹1,59,565.49)[8].

- **Eye-Pal Ace Plus**[9]

Eye-Pal Ace Plus, shown in figure 1.2, is a portable electronic text-to-speech device for people with visual impairment and for those who are blind. It is compatible with select refreshable Braille displays, allowing nearly instant conversion of text to Braille. Characterized by a contemporary design with time-tested simple controls, the Ace Plus has internal speakers, a rechargeable, user-replaceable battery, and a 10-inch built-in screen. The text is displayed in magnified font and read aloud in a natural-sounding voice.

The Ace Plus can be used as an electronic photo album, magnifier, alarm clock, and an appointment reminder as well. The Ace Plus has built-in WiFi capability to download Bookshare books and NFB Newsline publications and read them automatically. The device can also be used as a multi-tasking daily assistant for people with visual impairment and those who are blind.

It has a simple one-button email system and navigate your email with the simple controls or attach a voice message or audio labelled picture with the push of a button. There is also provision to magnify pages of books and publications or listen to them being read aloud. It is recommended by the American Foundation for the Blind (AFB)[10] and is very easy to use. It requires additional setup to hook-up to a computer and would require someone to help connect the wires everytime a digital copy need to be converted into audio. It costs about $1,330 (₹1,10,059)[11].

The Ace Plus has two 5-megapixel cameras that work simultaneously to take a snapshot of your material. Place the document in the landscape position, directly alongside the front of the Ace Plus. The optimal camera-viewable area is a letter/A4 paper sized (8.5 by 11 inches). Both cameras automatically provide sufficient lighting when taking a snapshot. No additional light is needed even in a dark room.Any material with printed text placed in front of the Ace Plus can be read out loud using the built-in speakers. Alternatively, headphones or a Braille display can be plugged in, or the Reading Mode can be set to Text Mode for those who prefer to read silently.

The top of the device has a panel with a set of Daisy arrow buttons on the left side, a large round button in the center, a volume knob on the right toward the back, and a half-moon shaped Scan Button. The front of the device has a Screen with a thumb wheel (Roller) at the top front of each side. When setting up the Ace Plus to scan a document, place the device on a flat surface with the front of the device facing you and the top panel facing the ceiling. It must be in this upright, flat position when scanning a document, but after it is scanned, you can hold the device comfortably on your lap, tilted however you would like while listening or reading.

Figure 1.2: Eye-Pal Ace Plus

1.3.1 LIMITATIONS OF EXISTING SYSTEMS

- The commercial equipments cost way too much and are out of reach for many students who are visually impaired or completely blind.
- They are also require additional hardware and software setup to be able to connect to a computer and read out the on-screen text and extract text from images.
- Many do-it-yourself (DIY) solutions that use OpenCV, Tesseract OCR and eSpeak in Raspberry Pi no longer work due to the updated software libraries deprecating some core features and newer versions of operating system switching to newer technologies.

1.4 PROPOSED SYSTEM

We will be using Raspberry Pi 3 Model B+, which is a micro-controller board. Then, a Raspberry Pi Camera module is used to capture images. Wired headset or a Bluetooth speaker is connected to hear the audio output from TTS engine. Visual output can be seen using a monitor (not necessary, just for reference to see processed result) connected to Pi through the HDMI port using cable.

The captured image is processed by Open Computer Vision (OpenCV), which is an open source computer vision library and image processing currently developed by Intel. OpenCV to make it easier for Optical Character Recognition (OCR) software and improve accuracy of text extraction. The OCR being used is Google's Tesseract OCR. The text extracted by Tesseract is then fed to Text-To-Speech module to convert the text into spoken words. eSpeakNG TTS engine is being used. Python programming is used to automate the entire process. The 'cv2', 'pytesseract' and 'espeakng' Python moduled str used for using OpenCV, Tesseract OCR and eSpeakNG TTS libraries from within Python program. Figure 1.3 shows the block diagram of proposed system.

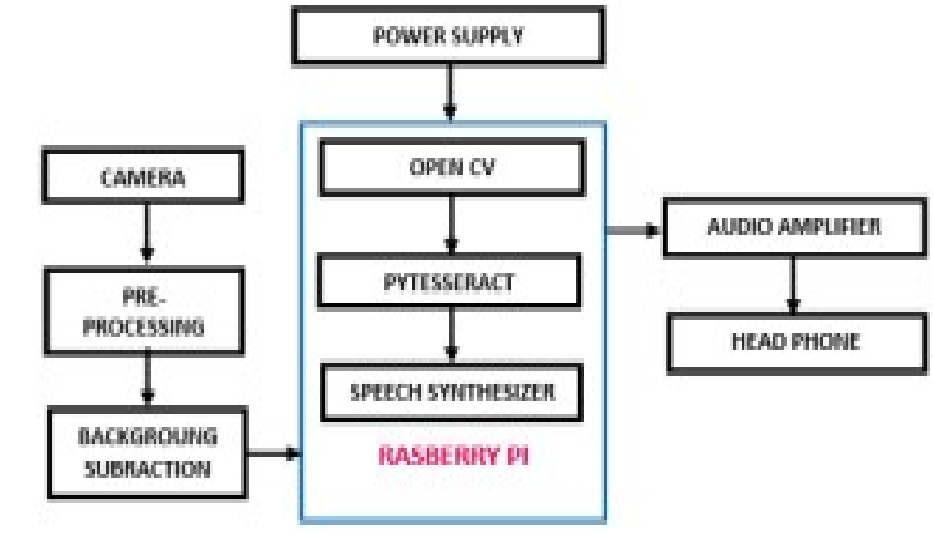


Figure 1.3: Block diagram of the proposed system

1.4.1 ADVANTAGES OF PROPOSED SYSTEM

- The device is portable and light weight.
- Can extract text from both on-screen active window and also images from camera containing texts.
- The entire process is automated and is done with a push-button press.
- No additional software or hardware setup is required.

CHAPTER 2
RASPBERRY PI

2.1 INTRODUCTION

Raspberry Pi is a series of small single-board computers (SBCs) developed in the United Kingdom by the Raspberry Pi Foundation in association with Broadcom. The Raspberry Pi project originally leaned towards the promotion of teaching basic computer science in schools and in developing countries. The original model became more popular than anticipated, selling outside its target market for uses such as robotics. It is widely used in many areas, such as for weather monitoring because of its low cost, modularity, and open design. It is typically used by computer and electronic hobbyists, due to its adoption of the HDMI and USB standards.

After the release of the second board type, the Raspberry Pi Foundation set up a new entity, named Raspberry Pi Trading, and installed Eben Upton as CEO, with the responsibility of developing technology. The Foundation was rededicated as an educational charity for promoting the teaching of basic computer science in schools and developing countries. Most Raspberry Pis are made in a Sony factory in Pencoed, Wales, while others are made in China and Japan. In 2015 the Raspberry Pi surpassed the ZX Spectrum in unit sales, becoming the best selling British computer.

2.2 HISTORY

The computer is inspired by Acorn's BBC Micro of 1981. The Model A, Model B and Model B+ names are references to the original models of the British educational BBC Micro computer, developed by Acorn Computers. According to Upton, the name "Raspberry Pi" was chosen with "Raspberry" as an ode to a tradition of naming early computer companies after fruit, and "Pi" as a reference to the Python programming language.

Fig 2.1: An early alpha-test board in operation

In 2006, early concepts of the Raspberry Pi were based on the Atmel ATmega644 microcontroller. Its schematics and PCB layout are publicly available. Figure 2.1 shows the early boards which were used for alpha-testing using different layout from later beta and production boards. Foundation trustee Eben Upton assembled a group of teachers, academics and computer enthusiasts to devise a computer to inspire children. The first ARM prototype version of the computer was mounted in a package the same size as a USB memory stick. It had a USB port on one end and an HDMI port on the other. The Foundation's goal was to offer two versions, priced at US$25 and $35. They started accepting orders for the higher priced Model B on 29 February 2012, the lower cost Model A on 4 February 2013 and the even lower cost (US$20) A+ on 10 November 2014. On 26 November 2015, the cheapest Raspberry Pi yet, the Raspberry Pi Zero, was launched at US$5 or £4. The launch timeline of the first generation Raspberry Pi is given below in a chronological order:

- 19 February 2012 – The first proof of concept SD card image that could be loaded onto an SD card to produce a preliminary operating system is released. The image was based on Debian 6.0 (Squeeze), with the LXDE desktop and the Midori browser, plus various programming tools. The image also runs on QEMU allowing the Raspberry Pi to be emulated on various other platforms.
- 29 February 2012 – Initial sales commence 29 February 2012 at 06:00 UTC. At the same time, it was announced that the model A, originally to have had 128 MB of RAM, was to be upgraded to 256 MB before release. The Foundation's website also announced: "Six years after the project's inception, we're nearly at the end of our first run of development – although it's just the beginning of the Raspberry Pi story." The web-shops of the two licensed manufacturers selling Raspberry Pi's

within the United Kingdom, Premier Farnell and RS Components, had their websites stalled by heavy web traffic immediately after the launch (RS Components briefly going down completely). Unconfirmed reports suggested that there were over two million expressions of interest or pre-orders. The official Raspberry Pi Twitter account reported that Premier Farnell sold out within a few minutes of the initial launch, while RS Components took over 100,000 pre orders on day one. Manufacturers were reported in March 2012 to be taking a "healthy number" of pre-orders.

- March 2012 – Shipping delays for the first batch were announced in March 2012, as the result of installation of an incorrect Ethernet port, but the Foundation expected that manufacturing quantities of future batches could be increased with little difficulty if required. "We have ensured we can get them [the Ethernet connectors with magnetics] in large numbers and Premier Farnell and RS Components [the two distributors] have been fantastic at helping to source components," Upton said. The first batch of 10,000 boards was manufactured in Taiwan and China.
- 8 March 2012 – Release Raspberry Pi Fedora Remix, the recommended Linux distribution, developed at Seneca College in Canada.
- March 2012 – The Debian port is initiated by Mike Thompson, former CTO of Atomz. The effort was largely carried out by Thompson and Peter Green, a volunteer Debian developer, with some support from the Foundation, who tested the resulting binaries that the two produced during the early stages (neither Thompson nor Green had physical access to the hardware, as boards were not widely accessible at the time due to demand). While the preliminary proof of concept image distributed by the Foundation before launch was also Debian-based, it differed from Thompson and Green's Raspbian effort in a couple of ways. The POC image was based on then-stable Debian Squeeze, while Raspbian aimed to track then-upcoming Debian Wheezy packages. Aside from the updated

packages that would come with the new release, Wheezy was also set to introduce the armhf architecture. The Squeeze-based POC image was limited to the armel architecture, which was, at the time of Squeeze's release, the latest attempt by the Debian project to have Debian run on the newest ARM embedded-application binary interface (EABI). The armhf architecture in Wheezy intended to make Debian run on the ARM VFP hardware floating-point unit, while armel was limited to emulating floating point operations in software. Since the Raspberry Pi included a VFP, being able to make use of the hardware unit would result in performance gains and reduced power use for floating point operations. The armhf effort in mainline Debian, however, was orthogonal to the work surrounding the Pi and only intended to allow Debian to run on ARMv7 at a minimum, which would mean the Pi, an ARMv6 device, would not benefit. As a result, Thompson and Green set out to build the 19,000 Debian packages for the device using a custom build cluster.

2.3 HARDWARE

The Raspberry Pi hardware has evolved through several versions that feature variations in the type of the central processing unit, amount of memory capacity, networking support, and peripheral-device support. Figure 2.2 describes block diagram of models B, B+, A and A+. In Model A, A+ and Pi Zero, the USB port is connected directly to the System-on-a-Chip (SoC).

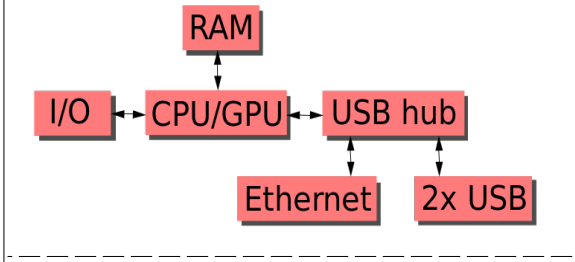


Fig 2.2: Block diagram describing models B, B+, A and A+

On the Pi 1 Model B+ and later models the USB/Ethernet chip contains a five-port USB hub, of which four ports are available, while the Pi 1 Model B only provides two. The Pi Zero models are similar, but lack the Ethernet and USB hub components. The Ethernet adapter is internally connected to an additional USB port, connected directly to

the SoC, but it uses a micro USB (OTG) port. Unlike all other Pi models, the 40 pin GPIO connector is omitted on the Pi Zero, with solderable through-holes only in the pin locations. The Pi Zero WH remedies this.

2.3.1 PROCESSOR

Processor speed ranges from 700 MHz to 1.4 GHz for the Pi 3 Model B+ or 1.5 GHz for the Pi 4; on-board memory ranges from 256 MB to 8 GB random-access memory (RAM), with only the Raspberry Pi 4 having more than 1 GB. Secure Digital (SD) cards in MicroSDHC form factor (SDHC on early models) are used to store the operating system and program memory, however some models also come with onboard eMMC storage and the Raspberry Pi 4 can also make use of USB-attached SSD storage for its operating system. The boards have one to five USB ports. For video output, HDMI and composite video are supported, with a standard 3.5 mm tip-ring-sleeve jack carrying mono audio together with composite video. Lower-level output is provided by a number of GPIO pins, which support common protocols like I²C. The B-models have an 8P8C Ethernet port and the Pi 3, Pi 4 and Pi Zero W have on-board Wi-Fi 802.11n and Bluetooth. The processor used in each generation and model of Raspberry Pi along with their specifications are given below:

- The First generation Raspberry Pi used the Broadcom BCM2835 SoC which includes a 700 MHz 32-bit ARM1176JZF-S processor, VideoCore IV Graphics Processing Unit (GPU) and RAM. It has a level 1 (L1) cache of 16 KB and a level 2 (L2) cache of 128 KB. The level 2 cache is used primarily by the GPU. The SoC is stacked underneath the RAM chip, so only its edge is visible. The ARM1176JZ(F)-S is the same CPU used in the original iPhone, although at a higher clock rate and was paired with a much faster GPU.
- Raspberry Pi 2's earlier V1.1 model used the Broadcom BCM2836 SoC with a 900 MHz 32-bit, quad-core ARM Cortex-A7 processor, with 256 KB shared L2 cache. The Raspberry Pi 2 V1.2 was upgraded to a Broadcom BCM2837 SoC

with a 1.2 GHz 64-bit quad-core ARM Cortex-A53 processor, the same one which is used on the Raspberry Pi 3 but underclocked to the same 900 MHz CPU clock speed as the V1.1. The BCM2836 SoC is no longer in production as of late 2016.

- Raspberry Pi 3 Model B uses a Broadcom BCM2837 SoC with a 1.2 GHz 64-bit quad-core ARM Cortex-A53 processor with 512 KB shared L2 cache. The Model A+ and B+ are clocked at 1.4 GHz.
- Raspberry Pi 4 uses the Broadcom BCM2711 SoC with a 1.5 GHz (later models are clocked at 1.8 GHz) 64-bit quad-core ARM Cortex-A72 processor, with 1 MB shared L2 cache. Unlike previous models, which all used a custom interrupt controller poorly suited for virtualisation, the interrupt controller on this SoC is compatible with the ARM Generic Interrupt Controller (GIC) architecture 2.0, providing hardware support for interrupt distribution when using ARM virtualisation capabilities.
- Raspberry Pi Zero and Zero W use the same Broadcom BCM2835 SoC as the first generation Raspberry Pi, although now running at 1 GHz CPU clock speed. The Raspberry Pi Zero W 2 uses the RP3A0-AU CPU, a 1 GHz 64 bit ARM Cortex A53 on 512MB of SDRAM. Documentation states this "system-on-package" is a Broadcom BCM2710A1 package, using a BCM2837 Broadcom chip as core, which is an ARM v8 quad-core.
- Raspberry Pi Pico uses the RP2040 running at 133 MHz.

2.3.2 RAM

The early designs of the Raspberry Pi Model A and B boards included only 256 MB of random access memory (RAM). Of this, the early beta Model B boards allocated 128 MB to the GPU by default, leaving only 128 MB for the CPU. On the early 256 MB releases of models A and B, three different splits were possible. The default split was 192 MB for the CPU, which should be sufficient for standalone 1080p video decoding, or for simple

3D processing. 224 MB was for Linux processing only, with only a 1080p framebuffer, and was likely to fail for any video or 3D. 128 MB was for heavy 3D processing, possibly also with video decoding. In comparison, the Nokia 701 uses 128 MB for the Broadcom VideoCore IV.

The later Model B with 512 MB RAM, was released on 15 October 2012 and was initially released with new standard memory split files (arm256_start.elf, arm384_start.elf, arm496_start.elf) with 256 MB, 384 MB, and 496 MB CPU RAM, and with 256 MB, 128 MB, and 16 MB video RAM, respectively. But about one week later, the foundation released a new version of start.elf that could read a new entry in config.txt (gpu_mem=*xx*) and could dynamically assign an amount of RAM (from 16 to 256 MB in 8 MB steps) to the GPU, obsoleting the older method of splitting memory, and a single start.elf worked the same for 256 MB and 512 MB Raspberry Pis.

The Raspberry Pi 2 and the Raspberry Pi 3 (B and B+ models) have 1 GB of RAM. The Raspberry Pi 3 Model A+ has 512 MB of RAM.

The Raspberry Pi 4 is available with 1, 2, 4 or 8 GB of RAM. A 1 GB model was originally available at launch in June 2019 but was discontinued in March 2020, and the 8 GB model was introduced in May 2020. The 1 GB model later returned in October 2021. The Raspberry Pi Zero and Zero W have 512 MB of RAM.

2.3.3 NETWORKING

The Model A, A+ and Pi Zero have no Ethernet circuitry and are commonly connected to a network using an external user-supplied USB Ethernet or Wi-Fi adapter. On the Model B and B+ the Ethernet port is provided by a built-in USB Ethernet adapter using the SMSC LAN9514 chip. The Raspberry Pi 3 and Pi Zero W (wireless) are equipped with 2.4 GHz WiFi 802.11n (150 Mbit/s) and Bluetooth 4.1 (24 Mbit/s) based on the Broadcom BCM43438 FullMAC chip with no official support for monitor mode (though it was implemented through unofficial firmware patching) and the Pi 3 also has a 10/100 Mbit/s Ethernet port. The Raspberry Pi 3B+ features dual-band IEEE 802.11b/g/n/ac

WiFi, Bluetooth 4.2, and Gigabit Ethernet (limited to approximately 300 Mbit/s by the USB 2.0 bus between it and the SoC). The Raspberry Pi 4 has full gigabit Ethernet (throughput is not limited as it is not funneled via the USB chip) .

2.3.4 PERIPHERALS

Although often pre-configured to operate as a headless computer, the Raspberry Pi may also optionally be operated with any generic USB computer keyboard and mouse. It may also be used with USB storage, USB to MIDI converters, and virtually any other device/component with USB capabilities, depending on the installed device drivers in the underlying operating system (many of which are included by default). Other peripherals can be attached through the various pins and connectors on the surface of the Raspberry Pi.

2.3.5 SPECIAL PURPOSE FEATURES

The RPi Zero, RPi1A, RPi3A+ and RPi4 can be used as a USB device or "USB gadget", plugged into another computer via a USB port on another machine. It can be configured in multiple ways, such as functioning as a serial or Ethernet device. Although originally requiring software patches, this was added into the mainline Raspbian distribution in May 2016.

Raspberry Pi models with a newer chipset can boot from USB mass storage, such as from a flash drive. Booting from USB mass storage is not available in the original Raspberry Pi models, the Raspberry Pi Zero, the Raspberry Pi Pico, the Raspberry Pi 2 A models, and the Raspberry Pi 2 B models with versions lower than 1.2.

2.4 SOFTWARE

2.4.1 OPERATING SYSTEMS

The Raspberry Pi Foundation provides Raspberry Pi OS (formerly called Raspbian), a Debian-based Linux distribution for download, as well as third-party Ubuntu, Windows

10 IoT Core, RISC OS, LibreELEC (specialised media centre distribution) and specialised distributions for the Kodi media centre and classroom management. It promotes Python and Scratch as the main programming languages, with support for many other languages. The default firmware is closed source, while unofficial open source is available. Many other operating systems can also run on the Raspberry Pi. The formally verified microkernel seL4 is also supported. There are several ways of installing multiple operating systems on one SD card.

The Raspberry Pi boards are capable of running the following independent operating systems:

- Broadcom VCOS – Proprietary operating system which includes an abstraction layer designed to integrate with existing kernels, such as ThreadX (which is used on the VideoCore4 processor), providing drivers and middleware for application development. In the case of the Raspberry Pi, this includes an application to start the ARM processor(s) and provide the publicly documented API over a mailbox interface, serving as its firmware. An incomplete source of a Linux port of VCOS is available as part of the reference graphics driver published by Broadcom.
- Haiku – an open source BeOS clone that has been compiled for the Raspberry Pi and several other ARM boards. Work on Pi 1 began in 2011, but only the Pi 2 will be supported.
- HelenOS – a portable microkernel-based multiserver operating system; has basic Raspberry Pi support since version 0.6.0
- Plan 9 from Bell Labs and Inferno(in beta)
- QNX
- RISC OS Pi (a special cut down version RISC OS Pico, for 16 MB cards and larger for all models of Pi 1 & 2, has also been made available.)
- Ultibo Core – OS-less unikerel Run Time Library based on Free Pascal. Lazarus IDE (Windows with 3rd party ports to Linux and MacOS). Most Pi models supported.

- Windows 10 IoT Core – a zero-price edition of Windows 10 offered by Microsoft that runs natively on the Raspberry Pi 2.

Linux-based Operating Systems that can run on Raspberry Pi:

- Android Things – an embedded version of the Android operating system designed for IoT device development. emteria.OS – an embedded, managed version of the Android operating system for professional fleet management.
- Alpine Linux – a Linux distribution based on musl and BusyBox, "designed for power users who appreciate security, simplicity and resource efficiency". PostMarketOS – distribution based on Alpine Linux, primarily developed for smartphones.
- Arch Linux ARM, a port of Arch Linux for ARM processors and Arch-based distribution Manjaro Linux ARM.
- Debian GNU/Linux-based distributions like Ubuntu, Kali Linux and RetroPie. Kali Linux is designed for digital forensics and penetration testing and RetroPie is an offshoot of Raspbian OS that uses Emulation Station as its front-end for RetroArch and other emulators like Mupen64 for retro gaming. Hardware like Freeplay tech can help replace Game boy internals with RetroPie emulation.
- Red Hat sponsored distributions – Fedora, CentOS (support Raspberry Pi 2 and later) and RedSleeve, a RHEL port. SUSE Linux Enterprise Server 12 SP2, Server 12 SP3 provide commercial support and OpenSUSE is the community focused distribution sponsored by SUSE.
- Slackware ARM – version 13.37 and later runs on the Raspberry Pi without modification. The 128–496 MB of available memory on the Raspberry Pi is at least twice the minimum requirement of 64 MB needed to run Slackware Linux on an ARM or i386 system. (Whereas the majority of Linux systems boot into a graphical user interface, Slackware's default user environment is the textual shell / command line interface.) The Fluxbox window manager running under the X

Window System requires an additional 48 MB of RAM.

- OpenWrt – a highly extensible Linux distribution for embedded devices (typically wireless routers). It supports Pi 1, 2, 3, 4 and Zero W.
- ArkOS – designed for website and email self-hosting.

BSD-based Operating Systems that can run on Raspberry Pi:

- FreeBSD
- NetBSD
- OpenBSD (only on 64-bit platforms, such as Raspberry Pi 3)

2.4.2 FIRMWARE

The official firmware is a freely redistributable binary blob, that is proprietary software. A minimal proof-of-concept open source firmware is also available, mainly aimed at initialising and starting the ARM cores as well as performing minimal startup that is required on the ARM side. It is also capable of booting a very minimal Linux kernel, with patches to remove the dependency on the mailbox interface being responsive. It is known to work on Raspberry Pi 1, 2 and 3, as well as some variants of Raspberry Pi Zero.

2.4.3 DRIVER APIS

Raspberry Pi can use a VideoCore IV GPU via a binary blob, which is loaded into the GPU at boot time from the SD-card, and additional software, that initially was closed source. This part of the driver code was later released. However, much of the actual driver work is done using the closed source GPU code. Application software makes calls to closed source run-time libraries (OpenMax, OpenGL ES or OpenVG), which in turn call an open source driver inside the Linux kernel, which then calls the closed source VideoCore IV GPU driver code. The API of the kernel driver is specific for these closed libraries. Video applications use OpenMAX, 3D applications use OpenGL ES and 2D applications use OpenVG, which both in turn use EGL. OpenMAX and EGL use the

open source kernel driver in turn.

The Raspberry Pi Foundation first announced it was working on a Vulkan driver in February 2020. A working Vulkan driver running Quake 3 at 100 frames per second on a 3B+ was revealed by a graphics engineer who had been working on it as a hobby project on 20 June. On 24 November 2020 Raspberry Pi Foundation announced that their driver for the Raspberry Pi 4 is Vulkan 1.0 conformant. Raspberry Pi Trading announced further driver conformance for Vulkan 1.1 and 1.2 on 26 October 2021 and 1 August 2022.

2.5 ACCESSORIES

1. Gertboard – A Raspberry Pi Foundation sanctioned device, designed for educational purposes, that expands the Raspberry Pi's GPIO pins to allow interface with and control of LEDs, switches, analogue signals, sensors and other devices. It also includes an optional Arduino compatible controller to interface with the Pi.
2. Camera – On 14 May 2013, the foundation and the distributors RS Components & Premier Farnell/Element 14 launched the Raspberry Pi camera board alongside a firmware update to accommodate it. The camera board is shipped with a flexible flat cable that plugs into the CSI connector which is located between the Ethernet and HDMI ports. In Raspbian, the user must enable the use of the camera board by running Raspi-config and selecting the camera option. The camera module costs €20 in Europe (9 September 2013). It uses the OmniVision OV5647 image sensor and can produce 1080p, 720p and 640x480p video. The dimensions are 25 mm × 20 mm × 9 mm. In May 2016, v2 of the camera was launched: it is an 8 megapixel camera using a Sony IMX219. In January 2023, v3 of the camera was launched: it is an 12 megapixel camera using a Sony IMX708. Figures 2.3 and 2.4 show the versions 1, 2 and 3 of Raspberry Pi camera module respectively.

Fig 2.3: Raspberry Pi Version 1 - 5 MegaPixel camera

Fig 2.4: 8 Megapixel Version 2 of the Pi Camera

3. Infrared Camera – In October 2013, the foundation announced that they would begin producing a camera module without an infrared filter, called the Pi NoIR.
4. High Quality Camera – In May 2020, the 12.3 megapixel Sony IMX477 Exmor sensor camera module was released with support for C- and CS-mount lenses. The unit initially retailed for US$50 with interchangeable lenses starting at US$25. Figure 2.5 shows a Raspberry Pi high quality camera module.

Fig 2.5: Raspberry Pi High Quality Camera Module

5. Official Display – On 8 September 2015, The foundation and the distributors RS Components & Premier Farnell/Element 14 launched the Raspberry Pi Touch

Display.

6. HAT (Hardware Attached on Top) expansion boards – Together with the Model B+, inspired by the Arduino shield boards, the interface for HAT boards was devised by the Raspberry Pi Foundation. Figure 2.6 shows a TV HAT attached on top of a Raspberry Pi 4 Model B. Each HAT board carries a small EEPROM (typically a CAT24C32WI-GT3) containing the relevant details of the board, so that the Raspberry Pi's OS is informed of the HAT, and the technical details of it, relevant to the OS using the HAT. Mechanical details of a HAT board, which uses the four mounting holes in their rectangular formation, are available online.

Fig 2.6: Raspberry Pi 4 Model B with a TV HAT card for DVB-T/T2 television reception attached

2.6 APPLICATIONS AND USES

Technology writer Glyn Moody described the project in May 2011 as a "potential BBC Micro 2.0", not by replacing PC compatible machines but by supplementing them. In March 2012 Stephen Pritchard echoed the BBC Micro successor sentiment in ITPRO. Alex Hope, co-author of the Next Gen report, is hopeful that the computer will engage children with the excitement of programming. Co-author Ian Livingstone suggested that the BBC could be involved in building support for the device, possibly branding it as the BBC Nano. The Centre for Computing History strongly supports the Raspberry Pi project, feeling that it could "usher in a new era". Before release, the board was showcased by ARM's CEO Warren East at an event in Cambridge outlining Google's ideas to improve UK science and technology education.

Harry Fairhead, however, suggests that more emphasis should be put on improving the educational software available on existing hardware, using tools such as Google App Inventor to return programming to schools, rather than adding new hardware

choices. Simon Rockman, writing in a ZDNet blog, was of the opinion that teens will have "better things to do", despite what happened in the 1980s.

In October 2012, the Raspberry Pi won T3's Innovation of the Year award, and futurist Mark Pesce cited a (borrowed) Raspberry Pi as the inspiration for his ambient device project MooresCloud. In October 2012, the British Computer Society reacted to the announcement of enhanced specifications by stating, "it's definitely something we'll want to sink our teeth into."

In June 2017, Raspberry Pi won the Royal Academy of Engineering MacRobert Award. The citation for the award to the Raspberry Pi said it was "for its inexpensive credit card-sized microcomputers, which are redefining how people engage with computing, inspiring students to learn coding and computer science and providing innovative control solutions for industry." Clusters of hundreds of Raspberry Pis have been used for testing programs destined for supercomputers.

2.6.1 COMMUNITY

The Raspberry Pi community was described by Jamie Ayre of FOSS software company AdaCore as one of the most exciting parts of the project. Community blogger Russell Davis said that the community strength allows the Foundation to concentrate on documentation and teaching. The community developed a fanzine around the platform called The MagPi which in 2015, was handed over to the Raspberry Pi Foundation by its volunteers to be continued in-house. A series of community Raspberry Jam events have been held across the UK and around the world.

2.6.2 EDUCATION

As of January 2012, enquiries about the board in the United Kingdom have been received from schools in both the state and private sectors, with around five times as much interest from the latter. It is hoped that businesses will sponsor purchases for less advantaged schools. The CEO of Premier Farnell said that the government of a country in the Middle

East has expressed interest in providing a board to every schoolgirl, to enhance her employment prospects.

In 2014, the Raspberry Pi Foundation hired a number of its community members including ex-teachers and software developers to launch a set of free learning resources for its website. The Foundation also started a teacher training course called Picademy with the aim of helping teachers prepare for teaching the new computing curriculum using the Raspberry Pi in the classroom.

Fig 2.7: NASA's Open Source Rover powered by a Raspberry Pi 3

In 2018, NASA launched the JPL Open Source Rover Project, which is a scaled down version of Curiosity rover and uses a Raspberry Pi as the control module, to encourage students and hobbyists to get involved in mechanical, software, electronics, and robotics engineering. This rover is shown in the figure 2.7 and it is using a Raspberry Pi 3 board as its brain.

2.6.3 HOME AUTOMATION

There are a number of developers and applications that are using the Raspberry Pi for home automation. These programmers are making an effort to modify the Raspberry Pi into a cost-affordable solution in energy monitoring and power consumption. Because of the relatively low cost of the Raspberry Pi, this has become a popular and economical alternative to the more expensive commercial solutions.

2.6.4 INDUSTRIAL AUTOMATION

In June 2014, Polish industrial automation manufacturer TECHBASE released ModBerry, an industrial computer based on the Raspberry Pi Compute Module. The device has a number of interfaces, most notably RS-485/232 serial ports, digital and

analogue inputs/outputs, CAN and economical 1-Wire buses, all of which are widely used in the automation industry. The design allows the use of the Compute Module in harsh industrial environments, leading to the conclusion that the Raspberry Pi is no longer limited to home and science projects, but can be widely used as an Industrial IoT solution and achieve goals of Industry 4.0.

In March 2018, SUSE announced commercial support for SUSE Linux Enterprise on the Raspberry Pi 3 Model B to support a number of undisclosed customers implementing industrial monitoring with the Raspberry Pi.

In January 2021, TECHBASE announced a Raspberry Pi Compute Module 4 cluster for AI accelerator, routing and file server use. The device contains one or more standard Raspberry Pi Compute Module 4s in an industrial DIN rail housing, with some versions containing one or more Coral Edge tensor processing units.

2.6.5 COMMERCIAL PRODUCTS

The Organelle is a portable synthesizer, a sampler, a sequencer, and an effects processor designed and assembled by Critter & Guitari. It incorporates a Raspberry Pi computer module running Linux.

OTTO is a digital camera created by Next Thing Co. It incorporates a Raspberry Pi Compute Module. It was successfully crowd-funded in a May 2014 Kickstarter campaign.

Slice is a digital media player which also uses a Compute Module as its heart. It was crowd-funded in an August 2014 Kickstarter campaign. The software running on Slice is based on Kodi. Numerous commercial thin client computer terminals use the Raspberry Pi.

AutoPi TMU device is a telematics unit which is built on top of a Raspberry Pi Compute Module 4 and incorporates the philosophy of which Raspberry Pi was built upon.

2.6.6 COVID-19 PANDEMIC

During the COVID-19 pandemic, demand increased primarily due to the increase in remote work, but also because of the use of many Raspberry Pi Zeros in ventilators for COVID-19 patients in countries such as Colombia, which were used to combat strain on the healthcare system. In March 2020, Raspberry Pi sales reached 640,000 units, the second largest month of sales in the company's history.

CHAPTER 3
OPEN SOURCE COMPUTER VISION

3.1 INTRODUCTION

OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library, originally developed by Intel. It was later supported by Willow Garage, then Itseez (which was later acquired by Intel). The library is cross-platform and licensed as free and open-source software under Apache License 2. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products. Being an Apache 2 licensed product, OpenCV makes it easy for businesses to utilize and modify the code.

OpenCV has C++, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and MacOS. OpenCV leans mostly towards real-time vision applications and takes advantage of MMX and SSE instructions when available. A full-featured CUDA and OpenCL interfaces are being actively developed right now. The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms and about 10 times as many functions that compose or support those algorithms. These algorithms can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects, produce 3D point clouds from stereo cameras, stitch images together to produce a high resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality, etc.

OpenCV is written natively in C++ and has a templated interface that works seamlessly with STL containers. It has more than 47 thousand people of user community

and estimated number of downloads exceeding 18 million. The library is used extensively in companies, research groups and by governmental bodies. Along with well-established companies like Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda, Toyota that employ the library, there are many start-ups such as Applied Minds, VideoSurf, and Zeitera, that make extensive use of OpenCV.

3.2 HISTORY

Officially launched in 1999 the OpenCV project was initially an Intel Research initiative to advance CPU-intensive applications, part of a series of projects including real-time ray tracing and 3D display walls. The main contributors to the project included a number of optimization experts in Intel Russia, as well as Intel's Performance Library Team. In the early days of OpenCV, the goals of the project were described as:

- Advance vision research by providing not only open but also optimized code for basic vision infrastructure. No more reinventing the wheel.
- Disseminate vision knowledge by providing a common infrastructure that developers could build on, so that code would be more readily readable and transferable.
- Advance vision-based commercial applications by making portable, performance-optimized code available for free – with a license that did not require code to be open or free itself.

1. The first alpha version of OpenCV was released to the public at the IEEE Conference on Computer Vision and Pattern Recognition in 2000, and five betas were released between 2001 and 2005. The first 1.0 version was released in 2006. A version 1.1 "pre-release" was released in October 2008.

2. The second major release of the OpenCV was in October 2009. OpenCV 2 includes major changes to the C++ interface, aiming at easier, more type-safe patterns, new functions, and better implementations for existing ones in terms of performance (especially on multi-core systems). Official releases now occur every six months and

development is now done by an independent Russian team supported by commercial corporations.

3. In August 2012, support for OpenCV was taken over by a non-profit foundation OpenCV.org, which maintains a developer and user site.

4. In May 2016, Intel signed an agreement to acquire Itseez, a leading developer of OpenCV.

5. In July 2020, OpenCV announced and began a Kickstarter campaign for the OpenCV AI Kit, a series of hardware modules and additions to OpenCV supporting Spatial AI.

3.3 FEATURES

OpenCV-Python is a library of Python bindings designed to solve computer vision problems. We first import `cv2` before using OpenCV functions inside a Python program. OpenCV has a lot of features and we cannot explain all of them here. So, a few of the most commonly used features will be explained in this section.

3.3.1 IMAGE RESIZING

OpenCV can scale images. Scaling comes in handy in many image processing as well as machine learning applications. It helps in reducing the number of pixels from an image and that has several advantages e.g. It can reduce the time of training of a neural network as the more the number of pixels in an image more is the number of input nodes that in turn increases the complexity of the model. It also helps in zooming in on images. Many times we need to resize the image i.e. either shrink it or scale it up to meet

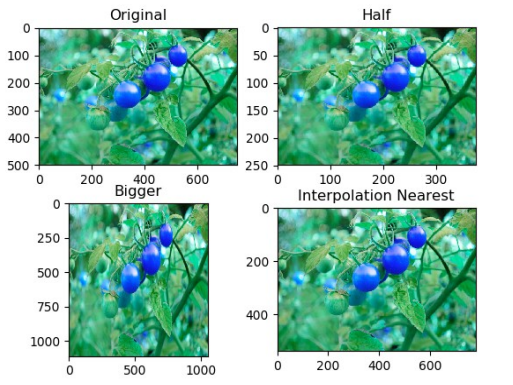


Figure 3.1: Various types of scaled images

the size requirements. OpenCV provides us several interpolation methods for resizing an image. The syntax is given below:

```
cv2.resize(source, dsize, destination, fx, fy, interpolation)
```

The parameters are explained below:

1. source: Input Image array (Single-channel, 8-bit or floating-point).
2. dsize: Size of the output array.
3. destination: Output array (Similar to the dimensions and type of Input image array). [optional]
4. fx: Scale factor along the horizontal axis. [optional]
5. fy: Scale factor along the vertical axis. [optional]
6. interpolation: one of cv2.INTER_AREA, cv2.INTER_CUBIC and cv2.INTER_LINEAR is used as the interpolation method.

An example image with its scaled output is shown in figure 3.1

3.3.2 CONVERTING IMAGE'S COLOR SPACE

The `cv2.cvtColor()` method is used to convert an image from one color space to another. There are more than 150 color-space conversion methods available in OpenCV. We are going to just focus on grayscaling an image and BGR to RGB color space conversions. The syntax is given below:

```
cv2.cvtColor(source, code[, destination[, dstCn]])
```

The parameters are explained below:

1. code: It is the color space conversion code.
2. dstCn: It is the number of channels in the destination image. If the parameter is 0 then the number of the channels is derived automatically from src and code. It is an optional parameter.

Grayscaling is the process of converting an image from other color spaces e.g. RGB, CMYK, HSV, etc. to shades of gray. It varies between complete black and complete white. Grayscaling an image is important because grayscale images are single-dimensional. For example, In RGB images there are three color channels and three dimensions while a monochromatic image has only single-dimensional as the colors are just a shade of gray.

There is also a reduction of model complexity. Consider training neural articles on RGB images of 10x10x3 pixels. The input layer will have 300 input nodes. On the other hand, the same neural network will need only 100 input nodes for grayscale images.

Grayscaling image is also needed for other algorithms to work. Many algorithms are customized to work only on grayscale images e.g. Canny edge detection function pre-implemented in the OpenCV library works on Grayscale images only. The syntax for converting an image to grayscale is shown below:

```
image = cv2.cvtColor(source, cv2.COLOR_RGB2GRAY )
cv2.imshow(output, image)
```

Figure 3.2: Image before conversion to grayscale

Figure 3.3: Converted grayscale image

Many images captured by cameras are in BGR format. BGR stands for Blue Green Red and as opposed to RGB (which stands for Red Green Blue), Red occupies the least significant area, Green the second, and Blue the third. For example, #FF0000 is pure

red when reading as an RGB hex color (#rrggbb), because the third area is FF – maximum value, full color and the other two areas are 00 – minimum value, no color. If #FF0000 were read as a BGR hex color, it'd be pure blue. The `dstCn` parameter for this operation is `cv2.COLOR_BGR2RGB`. The syntax is given below:

```
image = cv2.cvtColor(source, cv2.COLOR_BGR2RGB)
cv2.imshow(output, image)
```

The figure 3.4 is converted to BGR by OpenCV as shown in figure 3.5.

Figure 3.4: Captured BGR Image

Figure 3.5: OpenCV converted RGB Image

3.3.3 REMOVING NOISE IN AN IMAGE

The `cv2.GaussianBlur()` method is used to apply Gaussian smoothing on the input source image. The Gaussian blur is a type of image-blurring filter that uses a Gaussian function (which also expresses the normal distribution in statistics) for calculating the transformation to apply to each pixel in the image. The syntax is shown below:

```
blur = cv2.GaussianBlur(source, ksize, sigmaX[,
destination[, sigmaY[, borderType = BORDER_DEFAULT]]])
```

The parameters are explained below:

1. ksize: Gaussian Kernel Size. [height width]. height and width should be odd and

can have different values. If ksize is set to [0 0], then ksize is computed from sigma values.

2. sigmaX: Kernel standard deviation along X-axis (horizontal direction).
3. sigmaX: Kernel standard deviation along Y-axis (vertical direction). If sigmaY=0, then sigmaX value is taken for sigmaY

The `cv2.divide()` method is used for dividing the image by its blurred version, which is a background removal method. It whitens the background. We can remove noise in background by using this method. Before we use this method, the image is grayscaled as seen previously. The syntax is given below:

```
divide = cv2.divide(gray, blur, scale)
```

We can see this in effect in figure 3.7, when figure 3.6 is given as input image.

Figure 3.6: A noisy image

Figure 3.7: De-noised image

3.4 APPLICATIONS

OpenCV's application areas include:

- 2D and 3D feature toolkits
- Egomotion estimation
- Facial recognition system
- Gesture recognition
- Human–computer interaction (HCI)
- Mobile robotics
- Motion understanding

- Object detection
- Segmentation and recognition
- Stereopsis stereo vision: depth perception from 2 cameras
- Structure from motion (SFM)
- Motion video tracking
- Augmented reality

To support some of the above areas, OpenCV includes a statistical machine learning library that contains:

- Boosting
- Decision tree learning
- Gradient boosting trees
- Expectation-maximization algorithm
- k-nearest neighbor algorithm
- Naive Bayes classifier
- Artificial neural networks
- Random forest
- Support vector machine (SVM)
- Deep neural networks (DNN)

CHAPTER 4
OPTICAL CHARACTER RECOGNITION

4.1 INTRODUCTION

Optical character recognition or optical character reader (OCR) is the electronic or mechanical conversion of images of typed, handwritten or printed text into machine-encoded text, whether from a scanned document, a photo of a document, a scene-photo (for example the text on signs and billboards in a landscape photo) or from subtitle text superimposed on an image (for example: from a television broadcast). Widely used as a form of data entry from printed paper data records – whether passport documents, invoices, bank statements, computerized receipts, business cards, mail, printouts of static-data, or any suitable documentation – it is a common method of digitizing printed texts so that they can be electronically edited, searched, stored more compactly, displayed on-line, and used in machine processes such as cognitive computing, machine translation, (extracted) text-to-speech, key data and text mining. OCR is a field of research in pattern recognition, artificial intelligence and computer vision.

Early versions needed to be trained with images of each character, and worked on one font at a time. Advanced systems capable of producing a high degree of recognition accuracy for most fonts are now common, and with support for a variety of digital image file format inputs. Some systems are capable of reproducing formatted output that closely approximates the original page including images, columns, and other non-textual components.

4.2 HISTORY

Early optical character recognition may be traced to technologies involving telegraphy and creating reading devices for the blind. In 1914, Emanuel Goldberg developed a machine that read characters and converted them into standard telegraph code.

Concurrently, Edmund Fournier d'Albe developed the Optophone, a handheld scanner that when moved across a printed page, produced tones that corresponded to specific letters or characters.

In the late 1920s and into the 1930s, Emanuel Goldberg developed what he called a "Statistical Machine" for searching microfilm archives using an optical code recognition system. In 1931, he was granted USA Patent number 1,838,389 for the invention. The patent was acquired by IBM.

4.2.1 SOLUTIONS FOR BLIND AND VISUALLY IMPAIRED PEOPLE

In 1974, Ray Kurzweil started the company Kurzweil Computer Products, Inc. and continued development of omni-font OCR, which could recognize text printed in virtually any font (Kurzweil is often credited with inventing omni-font OCR, but it was in use by companies, including CompuScan, in the late 1960s and 1970s.) Kurzweil decided that the best application of this technology would be to create a reading machine for the blind, which would allow blind people to have a computer read text to them out loud. This device required the invention of two enabling technologies – the CCD flatbed scanner and the text-to-speech synthesizer. On January 13, 1976, the successful finished product was unveiled during a widely reported news conference headed by Kurzweil and the leaders of the National Federation of the Blind. In 1978, Kurzweil Computer Products began selling a commercial version of the optical character recognition computer program. LexisNexis was one of the first customers, and bought the program to upload legal paper and news documents onto its nascent online databases. Two years later, Kurzweil sold his company to Xerox, which had an interest in further commercializing paper-to-computer text conversion. Xerox eventually spun it off as Scansoft, which merged with Nuance Communications.

In the 2000s, OCR was made available online as a service (WebOCR), in a cloud computing environment, and in mobile applications like real-time translation of foreign-language signs on a smartphone. With the advent of smart-phones and smartglasses, OCR

can be used in internet connected mobile device applications that extract text captured using the device's camera. These devices that do not have OCR functionality built into the operating system will typically use an OCR API to extract the text from the image file captured and provided by the device. The OCR API returns the extracted text, along with information about the location of the detected text in the original image back to the device app for further processing (such as text-to-speech) or display.

Various commercial and open source OCR systems are available for most common writing systems, including Latin, Cyrillic, Arabic, Hebrew, Indic, Bengali (Bangla), Devanagari, Tamil, Telugu, Chinese, Japanese, and Korean characters.

4.3 TECHNIQUES

4.3.1 PRE-PROCESSING

OCR software often "pre-processes" images to improve the chances of successful recognition. Techniques include:

- De-skew – If the document was not aligned properly when scanned, it may need to be tilted a few degrees clockwise or counterclockwise in order to make lines of text perfectly horizontal or vertical.
- Despeckle – remove positive and negative spots, smoothing edges
- Binarisation – Convert an image from color or greyscale to black-and-white (called a "binary image" because there are two colors). The task of binarisation is performed as a simple way of separating the text (or any other desired image component) from the background. The task of binarisation itself is necessary since most commercial recognition algorithms work only on binary images since it proves to be simpler to do so. In addition, the effectiveness of the binarisation step influences to a significant extent the quality of the character recognition stage and the careful decisions are made in the choice of the binarisation employed for a given input image type; since the quality of the binarisation method employed to obtain the binary result depends on the type of the input image (scanned

document, scene text image, historical degraded document etc.).

- Line removal – Cleans up non-glyph boxes and lines
- Layout analysis or "zoning" – Identifies columns, paragraphs, captions, etc. as distinct blocks. Especially important in multi-column layouts and tables.
- Line and word detection – Establishes baseline for word and character shapes, separates words if necessary.
- Script recognition – In multilingual documents, the script may change at the level of the words and hence, identification of the script is necessary, before the right OCR can be invoked to handle the specific script.
- Character isolation or "segmentation" – For per-character OCR, multiple characters that are connected due to image artifacts must be separated; single characters that are broken into multiple pieces due to artifacts must be connected.
- Normalize aspect ratio and scale.

Segmentation of fixed-pitch fonts is accomplished relatively simply by aligning the image to a uniform grid based on where vertical grid lines will least often intersect black areas. For proportional fonts, more sophisticated techniques are needed because whitespace between letters can sometimes be greater than that between words, and vertical lines can intersect more than one character.

4.3.2 TEXT RECOGNITION

There are two basic types of core OCR algorithm, which may produce a ranked list of candidate characters.

- Matrix matching involves comparing an image to a stored glyph on a pixel-by-pixel basis; it is also known as "pattern matching", "pattern recognition", or "image correlation". This relies on the input glyph being correctly isolated from the rest of the image, and on the stored glyph being in a similar font and at the same scale. This technique works best with typewritten text and does not work well when new fonts are encountered. This is the technique the early physical photocell-based OCR implemented, rather directly.

- Feature extraction decomposes glyphs into "features" like lines, closed loops, line direction, and line intersections. The extraction features reduces the dimensionality of the representation and makes the recognition process computationally efficient. These features are compared with an abstract vector-like representation of a character, which might reduce to one or more glyph prototypes. General techniques of feature detection in computer vision are applicable to this type of OCR, which is commonly seen in "intelligent" handwriting recognition and indeed most modern OCR software. Nearest neighbour classifiers such as the k-nearest neighbors algorithm are used to compare image features with stored glyph features and choose the nearest match.

Software such as Cuneiform and Tesseract use a two-pass approach to character recognition. The second pass is known as "adaptive recognition" and uses the letter shapes recognized with high confidence on the first pass to recognize better the remaining letters on the second pass. This is advantageous for unusual fonts or low-quality scans where the font is distorted (e.g. blurred or faded). Modern OCR software include Google Docs OCR, ABBYY FineReader and Transym. Others like OCRopus and Tesseract uses neural networks which are trained to recognize whole lines of text instead of focusing on single characters.

A new technique known as iterative OCR automatically crops a document into sections based on page layout. OCR is performed on the sections individually using variable character confidence level thresholds to maximize page-level OCR accuracy. A patent from the United States Patent Office has been issued for this method. The OCR result can be stored in the standardized ALTO format, a dedicated XML schema maintained by the United States Library of Congress. Other common formats include hOCR and PAGE XML.

4.3.3 POST-PROCESSING

OCR accuracy can be increased if the output is constrained by a lexicon – a list of words

that are allowed to occur in a document. This might be, for example, all the words in the English language, or a more technical lexicon for a specific field. This technique can be problematic if the document contains words not in the lexicon, like proper nouns. Tesseract uses its dictionary to influence the character segmentation step, for improved accuracy.

The output stream may be a plain text stream or file of characters, but more sophisticated OCR systems can preserve the original layout of the page and produce, for example, an annotated PDF that includes both the original image of the page and a searchable textual representation.

"Near-neighbor analysis" can make use of co-occurrence frequencies to correct errors, by noting that certain words are often seen together. For example, "Washington, D.C." is generally far more common in English than "Washington DOC". Knowledge of the grammar of the language being scanned can also help determine if a word is likely to be a verb or a noun, for example, allowing greater accuracy. The Levenshtein Distance algorithm has also been used in OCR post-processing to further optimize results from an OCR API.

4.3.4 APPLICATION-SPECIFIC OPTIMIZATIONS

In recent years, the major OCR technology providers began to tweak OCR systems to deal more efficiently with specific types of input. Beyond an application-specific lexicon, better performance may be had by taking into account business rules, standard expression or rich information contained in color images. This strategy is called "Application-Oriented OCR" or "Customized OCR", and has been applied to OCR of license plates, invoices, screenshots, ID cards, driver licenses, and automobile manufacturing.

The New York Times has adapted the OCR technology into a proprietary tool they call - Document Helper, that enables their interactive news team to accelerate the processing of documents that need to be reviewed. They note that it enables them to process what amounts to as many as 5,400 pages per hour in preparation for reporters to

review the contents.

4.4 ACCURACY

Commissioned by the U.S. Department of Energy (DOE), the Information Science Research Institute (ISRI) had the mission to foster the improvement of automated technologies for understanding machine printed documents, and it conducted the most authoritative of the Annual Test of OCR Accuracy from 1992 to 1996. Recognition of Latin-script, typewritten text is still not 100% accurate even where clear imaging is available. One study based on recognition of 19th- and early 20th-century newspaper pages concluded that character-by-character OCR accuracy for commercial OCR software varied from 81% to 99%; total accuracy can be achieved by human review or Data Dictionary Authentication. Other areas—including recognition of hand printing, cursive handwriting, and printed text in other scripts (especially those East Asian language characters which have many strokes for a single character)—are still the subject of active research. The MNIST database is commonly used for testing systems' ability to recognise handwritten digits.

Accuracy rates can be measured in several ways, and how they are measured can greatly affect the reported accuracy rate. For example, if word context (basically a lexicon of words) is not used to correct software finding non-existent words, a character error rate of 1% (99% accuracy) may result in an error rate of 5% (95% accuracy) or worse if the measurement is based on whether each whole word was recognized with no incorrect letters. Using a large enough dataset is so important in a neural network based handwriting recognition solutions. On the other hand, producing natural datasets is very complicated and time-consuming. An example of the difficulties inherent in digitizing old text is the inability of OCR to differentiate between the "long s" and "f" characters. Web-based OCR systems for recognizing hand-printed text on the fly have become well known as commercial products in recent years (see Tablet PC history). Accuracy rates of 80% to 90% on neat, clean hand-printed characters can be achieved by pen computing

software, but that accuracy rate still translates to dozens of errors per page, making the technology useful only in very limited applications.

Recognition of cursive text is an active area of research, with recognition rates even lower than that of hand-printed text. Higher rates of recognition of general cursive script will likely not be possible without the use of contextual or grammatical information. For example, recognizing entire words from a dictionary is easier than trying to parse individual characters from script. Reading the *Amount* line of a cheque (which is always a written-out number) is an example where using a smaller dictionary can increase recognition rates greatly. The shapes of individual cursive characters themselves simply do not contain enough information to accurately (greater than 98%) recognize all handwritten cursive script. Most programs allow users to set "confidence rates". This means that if the software does not achieve their desired level of accuracy, a user can be notified for manual review. An error introduced by OCR scanning is sometimes termed a "scanno" (by analogy with the term "typo").

4.4.1 WORKAROUNDS

There are several techniques for solving the problem of character recognition by means other than improved OCR algorithms.

1. Forcing better input: special fonts like ocr-a, ocr-b, or micr fonts, with precisely specified sizing, spacing, and distinctive character shapes, allow a higher accuracy rate during transcription in bank check processing. Ironically, however, several prominent OCR engines were designed to capture text in popular fonts such as Arial or Times New Roman, and are incapable of capturing text in these fonts that are specialized and much different from popularly used fonts. As Google Tesseract can be trained to recognize new fonts, it can recognize OCR-A, OCR-B and MICR fonts. "Comb fields" are pre-printed boxes that encourage humans to write more legibly – one glyph per box. These are often printed in a "dropout color" which can be easily removed by the OCR system.

Palm OS used a special set of glyphs, known as "Graffiti" which are similar to printed English characters but simplified or modified for easier recognition on the platform's computationally limited hardware. Users would need to learn how to write these special glyphs. Zone-based OCR restricts the image to a specific part of a document. This is often referred to as "Template OCR".

2. Crowdsourcing: crowdsourcing humans to perform the character recognition can quickly process images like computer-driven ocr, but with higher accuracy for recognizing images than that obtained via computers. Practical systems include the Amazon Mechanical Turk and reCAPTCHA. The National Library of Finland has developed an online interface for users to correct OCRed texts in the standardized ALTO format. Crowd sourcing has also been used not to perform character recognition directly but to invite software developers to develop image processing algorithms, for example, through the use of rank-order tournaments.

4.5 TESSERACT OCR

Tesseract is an optical character recognition engine for various operating systems. It is free software, released under the Apache License. Originally developed by Hewlett-Packard as proprietary software in the 1980s, it was released as open source in 2005 and development has been sponsored by Google since 2006. In 2006, Tesseract was considered one of the most accurate open-source OCR engines available.

Tesseract has unicode (UTF-8) support, and can recognize more than 100 languages "out of the box" and supports various image formats including PNG, JPEG and TIFF. It also supports various output formats: plain text, hOCR (HTML), PDF, invisible-text-only PDF, TSV and ALTO (since version 4.1.0).

4.5.1 HISTORY

The Tesseract engine was originally developed as proprietary software at Hewlett Packard labs in Bristol, England and Greeley, Colorado between 1985 and 1994, with

more changes made in 1996 to port to Windows, and some migration from C to C++ in 1998. A lot of the code was written in C, and then some more was written in C++. Since then, all the code has been converted to at least compile with a C++ compiler. Very little work was done in the following decade. It was then released as open source in 2005 by Hewlett Packard and the University of Nevada, Las Vegas (UNLV). Tesseract development has been sponsored by Google since 2006. Version 4 adds LSTM based OCR engine and models for many additional languages and scripts, bringing the total to 116 languages. Additionally 37 scripts are supported. So it is for example possible to recognize text with a mix of Western and Central European languages by using the model for the Latin script it is written in.

Version 5 is the current stable of Tesseract OCR and was released on November 30, 2021 after more than two years of testing and developing. Newer minor versions and bugfix versions are available from GitHub. Latest source code is available from the main branch.

4.5.2 FEATURES

Tesseract was in the top three OCR engines in terms of character accuracy in 1995. It is available for Linux, Windows and Mac OS X. However, due to limited resources it is only rigorously tested by developers under Windows and Ubuntu.

Tesseract up to and including version 2 could only accept TIFF images of simple one-column text as inputs. These early versions did not include layout analysis, and so inputting multi-columned text, images, or equations produced garbled output. Since version 3.00 Tesseract has supported output text formatting, hOCR positional information and page-layout analysis. Support for a number of new image formats was added using the Leptonica library. Tesseract can detect whether text is monospaced or proportionally spaced.

The initial versions of Tesseract could only recognize English-language text. Tesseract v2 added six additional Western languages (French, Italian, German, Spanish,

Brazilian Portuguese, Dutch). Version 3 extended language support significantly to include ideographic (Chinese & Japanese) and right-to-left (e.g. Arabic, Hebrew) languages, as well as many more scripts. New languages included Arabic, Bulgarian, Catalan, Chinese (Simplified and Traditional), Croatian, Czech, Danish, German (Fraktur script), Greek, Finnish, Hebrew, Hindi, Hungarian, Indonesian, Japanese, Korean, Latvian, Lithuanian, Norwegian, Polish, Portuguese, Romanian, Russian, Serbian, Slovak (standard and Fraktur script), Slovenian, Swedish, Tagalog, Tamil, Thai, Turkish, Ukrainian and Vietnamese. V3.04, released in July 2015, added an additional 39 language/script combinations, bringing the total count of support languages to over 100. New language codes included: Amharic, Assamese, Azerbaijana in Cyrillic script, Tibetan, Bosnian, Cebuano, Welsh, Dzongkha, Persian, Irish, Gujarati, Haitian and Haitian Creole, Inuktitut, Javanese, Georgian and Old Georgian, Kazakh, Central Khmer, Kyrgyz, Kurdish, Lao, Latin, Marathi, Burmese, Nepali, Oriya, Punjabi, Pashto, Sanskrit, Sinhala, Serbian in Latin script, Syriac, Tajik, Tigrinya, Uyghur, Urdu, Uzbek, Uzbek in Cyrillic script and Yiddish. In addition, Tesseract can be trained to work in other languages.

Tesseract can process right-to-left text such as Arabic or Hebrew, many Indic scripts as well as CJK quite well. Accuracy rates are shown in this presentation for Tesseract tutorial at DAS 2016, Santorini by Ray Smith. It is suitable for use as a backend and can be used for more complicated OCR tasks including layout analysis by using a frontend such as OCRopus. Tesseract's output will have very poor quality if the input images are not preprocessed to suit it: Images (especially screenshots) must be scaled up such that the text x-height is at least 20 pixels, any rotation or skew must be corrected or no text will be recognized, low-frequency changes in brightness must be high-pass filtered, or

Figure 4.1: Example image with multi-lingual texts

Tesseract's binarization stage will destroy much of the page, and dark borders must be manually removed, or they will be misinterpreted as characters.

Multiple languages can be detected by using `-l [LANG]+[LANG]` flag in the command. Tesseract will only extract both the languages if both the language trained data packages are installed in the OS. For example, if the figure 4.1 is given as image input. The command to detect multiple languages along with the output shown in the terminal window after extracting text is shown below:

```
~ > tesseract <path_to_image>/<image_file_name> - -l eng+hin
Estimating resolution as 638
हिंदी से अंग्रेजी
HINDI TO
ENGLISH
```

Tesseract OCR also has the option to configure page segmentation, which can be controlled by using Page Segmentation Mode flag, by appending `--psm` to the command. All the Page Segmentation Modes available in the OCR are given below:

- Page segmentation modes:
  - 0 - Orientation and script detection (OSD) only.
  - 1 - Automatic page segmentation with OSD.
  - 2 - Automatic page segmentation, but no OSD or OCR.
  - 3 - Fully automatic page segmentation, but no OSD. (Default)
  - 4 - Assume a single column of text of variable sizes.
  - 5 - Assume a single uniform block of vertically aligned text.
  - 6 - Assume a single uniform block of text.
  - 7 - Treat the image as a single text line.
  - 8 - Treat the image as a single word.
  - 9 - Treat the image as a single word in a circle.
  - 10 - Treat the image as a single character.
  - 11 - Sparse text. Find as much text as possible in no particular order.

- 12 - Sparse text with OSD.
- 13 - Raw line. Treat the image as a single text line, bypassing hacks that are aimd at Tesseract

Tesseract OCR can also be configured to use specific text detection engine algorithm using the OCR Engine Modes flag, by appending `--oem` to the command. The OCR Engine Modes are given below:

- OCR Engine modes:
  - 0 - Legacy engine only.
  - 1 - Neural nets LSTM engine only.
  - 2 - Legacy + LSTM engines.
  - 3 Default, based on what is available.

Tesseract OCR also has the option to add user trained data which would be very useful for special cases and specific applications. There is option to add user pattern and word file trained by the user in the command as shown below:

```
tesseract <path_to_image>/<image_file_name> --user-words <path> --user-patterns <path>
```

4.6 APPLICATIONS

OCR engines have been developed into many kinds of domain-specific OCR applications, such as receipt OCR, invoice OCR, check OCR, legal billing document OCR.

They can be used for:

- Data entry for business documents, e.g. Cheque, passport, invoice, bank statement and receipt.
- Automatic number plate recognition.
- In airports, for passport recognition and information extraction.
- Automatic insurance documents key information extraction.
- Traffic-sign recognition.

- Extracting business card information into a contact list.
- More quickly make textual versions of printed documents, e.g. book scanning for Project Gutenberg
- Make electronic images of printed documents searchable, e.g. Google Books.
- Converting handwriting in real-time to control a computer (pen computing).
- Defeating CAPTCHA anti-bot systems, though these are specifically designed to prevent OCR. The purpose can also be to test the robustness of CAPTCHA anti-bot systems.
- Assistive technology for blind and visually impaired users.
- Writing the instructions for vehicles by identifying CAD images in a database that are appropriate to the vehicle design as it changes in real time.
- Making scanned documents searchable by converting them to searchable PDFs.

CHAPTER 5
SPEECH SYNTHESIS

5.1 INTRODUCTION

Speech synthesis is the artificial production of human speech. A computer system used for this purpose is called a speech synthesizer, and can be implemented in software or hardware products. A text-to-speech (TTS) system converts normal language text into speech; other systems render symbolic linguistic representations like phonetic transcriptions into speech. The reverse process is speech recognition. Figure 5.1 shows a overview of a typical Text-To-Speech (TTS) system.

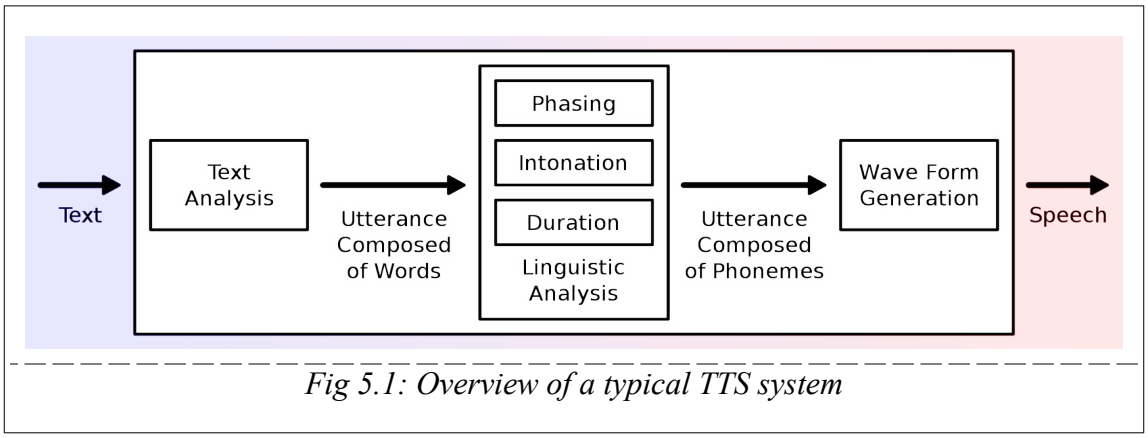


Fig 5.1: Overview of a typical TTS system

stored in a database. Systems differ in the size of the stored speech units; a system that stores phones or diphones provides the largest output range, but may lack clarity. For specific usage domains, the storage of entire words or sentences allows for high-quality output. Alternatively, a synthesizer can incorporate a model of the vocal tract and other human voice characteristics to create a completely "synthetic" voice output. The quality of a speech synthesizer is judged by its similarity to the human voice and by its ability to be understood clearly. An intelligible text-to-speech program allows people with visual impairments or reading disabilities to listen to written words on a home computer. Many computer operating systems have included speech synthesizers since the early 1990s.

A text-to-speech system (or "engine") is composed of two parts: a front-end and a

back-end. The front-end has two major tasks. First, it converts raw text containing symbols like numbers and abbreviations into the equivalent of written-out words. This process is often called text normalization, pre-processing, or tokenization. The front-end then assigns phonetic transcriptions to each word, and divides and marks the text into prosodic units, like phrases, clauses, and sentences. The process of assigning phonetic transcriptions to words is called text-to-phoneme or grapheme-to-phoneme conversion. Phonetic transcriptions and prosody information together make up the symbolic linguistic representation that is output by the front-end. The back-end—often referred to as the synthesizer—then converts the symbolic linguistic representation into sound. In certain systems, this part includes the computation of the target prosody (pitch contour, phoneme durations), which is then imposed on the output speech.

5.2 HISTORY

Long before the invention of electronic signal processing, some people tried to build machines to emulate human speech. Some early legends of the existence of "Brazen Heads" involved Pope Silvester II (d. 1003 AD), Albertus Magnus (1198–1280), and Roger Bacon (1214–1294). In 1779 the German-Danish scientist Christian Gottlieb Kratzenstein won the first prize in a competition announced by the Russian Imperial Academy of Sciences and Arts for models he built of the human vocal tract that could produce the five long vowel sounds (in International Phonetic Alphabet notation: [aː], [eː], [iː], [oː] and [uː]). There followed the bellows-operated "acoustic-mechanical speech machine" of Wolfgang von Kempelen of Pressburg, Hungary, described in a 1791 paper. This machine added models of the tongue and lips, enabling it to produce consonants as well as vowels. In 1837, Charles Wheatstone produced a "speaking machine" based on von Kempelen's design, and in 1846, Joseph Faber exhibited the "Euphonia". In 1923 Paget resurrected Wheatstone's design.

In the 1930s Bell Labs developed the vocoder, which automatically analyzed speech into its fundamental tones and resonances. From his work on the vocoder, Homer

Dudley developed a keyboard-operated voice-synthesizer called The Voder (Voice Demonstrator), which he exhibited at the 1939 New York World's Fair.

Dr. Franklin S. Cooper and his colleagues at Haskins Laboratories built the Pattern playback in the late 1940s and completed it in 1950. There were several different versions of this hardware device; only one currently survives. The machine converts pictures of the acoustic patterns of speech in the form of a spectrogram back into sound. Using this device, Alvin Liberman and colleagues discovered acoustic cues for the perception of phonetic segments (consonants and vowels).

5.2.1 ELECTRONIC DEVICES WITH SPEECH SYNTHESIS COPABILITY

The first computer-based speech-synthesis systems originated in the late 1950s. Noriko Umeda et al. developed the first general English text-to-speech system in 1968, at the Electrotechnical Laboratory in Japan. In 1961, physicist John Larry Kelly, Jr and his colleague Louis Gerstman used an IBM 704 computer to synthesize speech, an event among the most prominent in the history of Bell Labs. Kelly's voice recorder synthesizer (vocoder) recreated the song "Daisy Bell", with musical accompaniment from Max Mathews. Coincidentally, Arthur C. Clarke was visiting his friend and colleague John Pierce at the Bell Labs Murray Hill facility. Clarke was so impressed by the demonstration that he used it in the climactic scene of his screenplay for his novel 2001: A Space Odyssey, where the HAL 9000 computer sings the same song as astronaut Dave Bowman puts it to sleep. Despite the success of purely electronic speech synthesis, research into mechanical speech-synthesizers continues.

Linear predictive coding (LPC), a form of speech coding, began development with the work of Fumitada Itakura of Nagoya University and Shuzo Saito of Nippon Telegraph and Telephone (NTT) in 1966. Further developments in LPC technology were made by Bishnu S. Atal and Manfred R. Schroeder at Bell Labs during the 1970s. LPC was later the basis for early speech synthesizer chips, such as the Texas Instruments LPC Speech Chips used in the Speak & Spell toys from 1978.

In 1975, Fumitada Itakura developed the line spectral pairs (LSP) method for high-compression speech coding, while at NTT. From 1975 to 1981, Itakura studied problems in speech analysis and synthesis based on the LSP method. In 1980, his team developed an LSP-based speech synthesizer chip. LSP is an important technology for speech synthesis and coding, and in the 1990s was adopted by almost all international speech coding standards as an essential component, contributing to the enhancement of digital speech communication over mobile channels and the internet.

In 1975, MUSA was released, and was one of the first Speech Synthesis systems. It consisted of a stand-alone computer hardware and a specialized software that enabled it to read Italian. A second version, released in 1978, was also able to sing Italian in an "a cappella" style.

Dominant systems in the 1980s and 1990s were the DECtalk system, based largely on the work of Dennis Klatt at MIT, and the Bell Labs system; the latter was one of the first multilingual language-independent systems, making extensive use of natural language processing methods.

Handheld electronics featuring speech synthesis began emerging in the 1970s. One of the first was the Telesensory Systems Inc. (TSI) Speech+ portable calculator for the blind in 1976. Other devices had primarily educational purposes, such as the Speak & Spell toy produced by Texas Instruments in 1978.

Fidelity released a speaking version of its electronic chess computer in 1979 which is shown in Figure 5.2. The first video game to feature speech synthesis was the 1980 shoot 'em up arcade game, Stratovox (known in Japan as Speak & Rescue), from Sun Electronics. The first personal computer game with speech synthesis was Manbiki Shoujo

Fig 5.2: Fidelity Voice Chess Challenger (1979), the first talking chess computer

(Shoplifting Girl), released in 1980 for the PET 2001, for which the game's developer, Hiroshi Suzuki, developed a "zero cross" programming technique to produce a synthesized speech waveform. Another early example, the arcade version of Berzerk, also dates from 1980. The Milton Bradley Company produced the first multi-player electronic game using voice synthesis, Milton, in the same year.

Early electronic speech-synthesizers sounded robotic and were often barely intelligible. The quality of synthesized speech has steadily improved, but as of 2016 output from contemporary speech synthesis systems remains clearly distinguishable from actual human speech. Synthesized voices typically sounded male until 1990, when Ann Syrdal, at AT&T Bell Laboratories, created a female voice. Kurzweil predicted in 2005 that as the cost-performance ratio caused speech synthesizers to become cheaper and more accessible, more people would benefit from the use of text-to-speech programs.

5.3 SYNTHESIZER TECHNOLOGIES

The most important qualities of a speech synthesis system are naturalness and intelligibility. Naturalness describes how closely the output sounds like human speech, while intelligibility is the ease with which the output is understood. The ideal speech synthesizer is both natural and intelligible. Speech synthesis systems usually try to maximize both characteristics. The two primary technologies generating synthetic speech waveforms are concatenative synthesis and formant synthesis. Each technology has strengths and weaknesses, and the intended uses of a synthesis system will typically determine which approach is used.

5.3.1 CONCATENATION SYNTHESIS

Concatenative synthesis is based on the concatenation (stringing together) of segments of recorded speech. Generally, concatenative synthesis produces the most natural-sounding synthesized speech. However, differences between natural variations in speech and the nature of the automated techniques for segmenting the waveforms sometimes result in

audible glitches in the output. There are three main sub-types of concatenative synthesis:

1. Unit Selection Synthesis:

Unit selection synthesis uses large databases of recorded speech. During database creation, each recorded utterance is segmented into some or all of the following: individual phones, diphones, half-phones, syllables, morphemes, words, phrases, and sentences. Typically, the division into segments is done using a specially modified speech recognizer set to a "forced alignment" mode with some manual correction afterward, using visual representations such as the waveform and spectrogram. An index of the units in the speech database is then created based on the segmentation and acoustic parameters like the fundamental frequency (pitch), duration, position in the syllable, and neighboring phones. At run time, the desired target utterance is created by determining the best chain of candidate units from the database (unit selection). This process is typically achieved using a specially weighted decision tree.

Unit selection provides the greatest naturalness, because it applies only a small amount of digital signal processing (DSP) to the recorded speech. DSP often makes recorded speech sound less natural, although some systems use a small amount of signal processing at the point of concatenation to smooth the waveform. The output from the best unit-selection systems is often indistinguishable from real human voices, especially in contexts for which the TTS system has been tuned. However, maximum naturalness typically require unit-selection speech databases to be very large, in some systems ranging into the gigabytes of recorded data, representing dozens of hours of speech. Also, unit selection algorithms have been known to select segments from a place that results in less than ideal synthesis (e.g. minor words become unclear) even when a better choice exists in the database. Recently, researchers have proposed various automated methods to detect unnatural segments in unit-selection speech synthesis systems.

2. Diphone Synthesis:

Diphone synthesis uses a minimal speech database containing all the diphones (sound-to-sound transitions) occurring in a language. The number of diphones depends on the phonotactics of the language: for example, Spanish has about 800 diphones, and German about 2500. In diphone synthesis, only one example of each diphone is contained in the speech database. At runtime, the target prosody of a sentence is superimposed on these minimal units by means of digital signal processing techniques such as linear predictive coding, PSOLA or MBROLA. or more recent techniques such as pitch modification in the source domain using discrete cosine transform. Diphone synthesis suffers from the sonic glitches of concatenative synthesis and the robotic-sounding nature of formant synthesis, and has few of the advantages of either approach other than small size. As such, its use in commercial applications is declining, although it continues to be used in research because there are a number of freely available software implementations. An early example of Diphone synthesis is a teaching robot, Leachim, that was invented by Michael J. Freeman. Leachim contained information regarding class curricular and certain biographical information about the students whom it was programmed to teach. It was tested in a fourth grade classroom in the Bronx, New York.

3. Domain-Specific Synthesis:

Domain-specific synthesis concatenates prerecorded words and phrases to create complete utterances. It is used in applications where the variety of texts the system will output is limited to a particular domain, like transit schedule announcements or weather reports. The technology is very simple to implement, and has been in commercial use for a long time, in devices like talking clocks and calculators. The level of naturalness of these systems can be very high because the variety of sentence types is limited, and they closely match the prosody and intonation of the original recordings.

Because these systems are limited by the words and phrases in their databases, they are not general-purpose and can only synthesize the combinations of words and

phrases with which they have been pre-programmed. The blending of words within naturally spoken language however can still cause problems unless the many variations are taken into account. For example, in non-rhotic dialects of English the "r" in words like "clear" is usually only pronounced when the following word has a vowel as its first letter. Likewise in French, many final consonants become no longer silent if followed by a word that begins with a vowel, an effect called liaison. This alternation cannot be reproduced by a simple word-concatenation system, which would require additional complexity to be context-sensitive.

5.3.2 FORMANT SYNTHESIS

Formant synthesis does not use human speech samples at runtime. Instead, the synthesized speech output is created using additive synthesis and an acoustic model (physical modelling synthesis). Parameters such as fundamental frequency, voicing, and noise levels are varied over time to create a waveform of artificial speech. This method is sometimes called rules-based synthesis; however, many concatenative systems also have rules-based components. Many systems based on formant synthesis technology generate artificial, robotic-sounding speech that would never be mistaken for human speech. However, maximum naturalness is not always the goal of a speech synthesis system, and formant synthesis systems have advantages over concatenative systems. Formant-synthesized speech can be reliably intelligible, even at very high speeds, avoiding the acoustic glitches that commonly plague concatenative systems. High-speed synthesized speech is used by the visually impaired to quickly navigate computers using a screen reader. Formant synthesizers are usually smaller programs than concatenative systems because they do not have a database of speech samples. They can therefore be used in embedded systems, where memory and microprocessor power are especially limited. Because formant-based systems have complete control of all aspects of the output speech, a wide variety of prosodies and intonations can be output, conveying not just questions and statements, but a variety of emotions and tones of voice.

Examples of non-real-time but highly accurate intonation control in formant synthesis include the work done in the late 1970s for the Texas Instruments toy Speak & Spell, and in the early 1980s Sega arcade machines and in many Atari, Inc. arcade games using the TMS5220 LPC Chips. Creating proper intonation for these projects was painstaking, and the results have yet to be matched by real-time text-to-speech interfaces.

5.4 SPEECH SYNTHESIS CHALLENGES

5.4.1 TEXT NORMALIZATION

The process of normalizing text is rarely straightforward. Texts are full of heteronyms, numbers, and abbreviations that all require expansion into a phonetic representation. There are many spellings in English which are pronounced differently based on context. For example, "My latest project is to learn how to better project my voice" contains two pronunciations of "project".

Most text-to-speech (TTS) systems do not generate semantic representations of their input texts, as processes for doing so are unreliable, poorly understood, and computationally ineffective. As a result, various heuristic techniques are used to guess the proper way to disambiguate homographs, like examining neighboring words and using statistics about frequency of occurrence.

Recently TTS systems have begun to use HMMs (discussed above) to generate "parts of speech" to aid in disambiguating homographs. This technique is quite successful for many cases such as whether "read" should be pronounced as "red" implying past tense, or as "reed" implying present tense. Typical error rates when using HMMs in this fashion are usually below five percent. These techniques also work well for most European languages, although access to required training corpora is frequently difficult in these languages.

Deciding how to convert numbers is another problem that TTS systems have to address. It is a simple programming challenge to convert a number into words (at least in English), like "1325" becoming "one thousand three hundred twenty-five". However,

numbers occur in many different contexts; "1325" may also be read as "one three two five", "thirteen twenty-five" or "thirteen hundred and twenty five". A TTS system can often infer how to expand a number based on surrounding words, numbers, and punctuation, and sometimes the system provides a way to specify the context if it is ambiguous. Roman numerals can also be read differently depending on context. For example, "Henry VIII" reads as "Henry the Eighth", while "Chapter VIII" reads as "Chapter Eight".

Similarly, abbreviations can be ambiguous. For example, the abbreviation "in" for "inches" must be differentiated from the word "in", and the address "12 St John St." uses the same abbreviation for both "Saint" and "Street". TTS systems with intelligent front ends can make educated guesses about ambiguous abbreviations, while others provide the same result in all cases, resulting in nonsensical (and sometimes comical) outputs, such as "Ulysses S. Grant" being rendered as "Ulysses South Grant".

5.4.2 TEXT-TO-PHONEME

Speech synthesis systems use two basic approaches to determine the pronunciation of a word based on its spelling, a process which is often called text-to-phoneme or grapheme-to-phoneme conversion (phoneme is the term used by linguists to describe distinctive sounds in a language). The simplest approach to text-to-phoneme conversion is the dictionary-based approach, where a large dictionary containing all the words of a language and their correct pronunciations is stored by the program. Determining the correct pronunciation of each word is a matter of looking up each word in the dictionary and replacing the spelling with the pronunciation specified in the dictionary. The other approach is rule-based, in which pronunciation rules are applied to words to determine their pronunciations based on their spellings. This is similar to the "sounding out", or synthetic phonics, approach to learning reading.

Each approach has advantages and drawbacks. The dictionary-based approach is quick and accurate, but completely fails if it is given a word which is not in its dictionary.

As dictionary size grows, so too does the memory space requirements of the synthesis system. On the other hand, the rule-based approach works on any input, but the complexity of the rules grows substantially as the system takes into account irregular spellings or pronunciations. (Consider that the word "of" is very common in English, yet is the only word in which the letter "f" is pronounced [v].) As a result, nearly all speech synthesis systems use a combination of these approaches.

Languages with a phonemic orthography have a very regular writing system, and the prediction of the pronunciation of words based on their spellings is quite successful. Speech synthesis systems for such languages often use the rule-based method extensively, resorting to dictionaries only for those few words, like foreign names and loanwords, whose pronunciations are not obvious from their spellings. On the other hand, speech synthesis systems for languages like English, which have extremely irregular spelling systems, are more likely to rely on dictionaries, and to use rule-based methods only for unusual words, or words that aren't in their dictionaries.

5.4.3 EVALUATION

The consistent evaluation of speech synthesis systems may be difficult because of a lack of universally agreed objective evaluation criteria. Different organizations often use different speech data. The quality of speech synthesis systems also depends on the quality of the production technique (which may involve analogue or digital recording) and on the facilities used to replay the speech. Evaluating speech synthesis systems has therefore often been compromised by differences between production techniques and replay facilities.

Since 2005, however, some researchers have started to evaluate speech synthesis systems using a common speech dataset.

5.4.4 PROSODICS AND EMOTIONAL CONTENT

A study in the journal Speech Communication by Amy Drahota and colleagues at the University of Portsmouth, UK, reported that listeners to voice recordings could determine, at better than chance levels, whether or not the speaker was smiling. It was suggested that identification of the vocal features that signal emotional content may be used to help make synthesized speech sound more natural. One of the related issues is modification of the pitch contour of the sentence, depending upon whether it is an affirmative, interrogative or exclamatory sentence. One of the techniques for pitch modification uses discrete cosine transform in the source domain (linear prediction residual). Such pitch synchronous pitch modification techniques need a priori pitch marking of the synthesis speech database using techniques such as epoch extraction using dynamic plosion index applied on the integrated linear prediction residual of the voiced regions of speech.

5.5 ESPEAKNG TTS

eSpeakNG is a free and open-source, cross-platform, compact, software speech synthesizer. It uses a formant synthesis method, providing many languages in a relatively small file size. eSpeakNG (Next Generation) is a continuation of the original developer's project with more feedback from native speakers.

Because of its small size and many languages, eSpeakNG is included in NVDA open source screen reader for Windows, as well as Android, Ubuntu and other Linux distributions. Its predecessor eSpeak was recommended by Microsoft in 2016 and was used by Google Translate for 27 languages in 2010; 17 of these were subsequently replaced by proprietary voices. The quality of the language voices varies greatly. In eSpeakNG's predecessor eSpeak, the initial versions of some languages were based on information found on Wikipedia. Some languages have had more work or feedback from native speakers than others. Most of the people who have helped to improve the various languages are blind users of text-to-speech.

5.5.1 HISTORY

The conception and history of eSpeak and eSpeakNG is given below in a chronological order:

- In 1995, Jonathan Duddington released the Speak speech synthesizer for RISC OS computers supporting British English. On 17 February 2006, Speak 1.05 was released under the GPLv2 license, initially for Linux, with a Windows SAPI 5 version added in January 2007. Development on Speak continued until version 1.14, when it was renamed to eSpeak.
- Development of eSpeak continued from 1.16 (there was not a 1.15 release) with the addition of an eSpeakEdit program for editing and building the eSpeak voice data. These were only available as separate source and binary downloads up to eSpeak 1.24. The 1.24.02 version of eSpeak was the first version of eSpeak to be version controlled using subversion, with separate source and binary downloads made available on SourceForge. From eSpeak 1.27, eSpeak was updated to use the GPLv3 license. The last official eSpeak release was 1.48.04 for Windows and Linux, 1.47.06 for RISC OS and 1.45.04 for macOS. The last development release of eSpeak was 1.48.15 on 16 April 2015.
- eSpeak uses the Usenet scheme to represent phonemes with ASCII characters.
- On 25 June 2010, Reece Dunn started a fork of eSpeak on GitHub using the 1.43.46 release. This started off as an effort to make it easier to build eSpeak on Linux and other POSIX platforms.
- On 4 October 2015 (6 months after the 1.48.15 release of eSpeak), this fork started diverging more significantly from the original eSpeak.
- On 8 December 2015, there were discussions on the eSpeak mailing list about the lack of activity from Jonathan Duddington over the previous 8 months from the last eSpeak development release. This evolved into discussions of continuing development of eSpeak in Jonathan's absence. The result of this was the creation of the espeak-ng (Next Generation) fork, using the GitHub version of eSpeak as

the basis for future development.

- On 11 December 2015, the espeak-ng fork was started. The first release of espeak-ng was 1.49.0 on 10 September 2016, containing significant code cleanup, bug fixes, and language updates.

5.5.2 FEATURES

- eSpeakNG can be used as a command-line program, or as a shared library.
- It supports Speech Synthesis Markup Language (SSML).
- Language voices are identified by the language's ISO 639-1 code. They can be modified by "voice variants". These are text files which can change characteristics such as pitch range, add effects such as echo, whisper and croaky voice, or make systematic adjustments to formant frequencies to change the sound of the voice. For example, "af" is the Afrikaans voice. "af+f2" is the Afrikaans voice modified with the "f2" voice variant.
- eSpeakNG uses an ASCII representation of phoneme names which is loosely based on the Usenet system. Phonetic representations can be included within text input by including them within double square-brackets.
- For example: espeak-ng -v en "Hello World" will say Hello world in English.

The following are the command-line options and flags that can be appended after the `espeak-ng` command in a terminal emulator:

- -f <text file>: Text file to speak.
- --stdin: Read text input from stdin till to the end of a stream at once.
- -d <device>: Use the specified device to speak the audio on. If not specified, the default audio device is used.
- -q: Quiet, don't produce any speech (may be useful with -x).
- -a <integer>: Amplitude, 0 to 200, default is 100.
- -g <integer>: Word gap. Pause between words, units of 10ms at the default speed.
- -k <integer>: Indicate capital letters with: 1=sound, 2=the word "capitals", higher

values = a pitch increase (try -k20)

- .-l <integer>: Line length. If not zero (which is the default), consider lines less than this length as end-of-clause.
- -p <integer>: Pitch adjustment, 0 to 99, default is 50.
- -s <integer>: Speed in words per minute, default is 175.
- -v <voice name>: Use voice file of this name from espeak-ng-data/voices. A variant can be specified using +, such as af+m3.
- -w <wave file name>: Write output to this WAV file, rather than speaking it directly.
- --split=<minutes>: Used with -w to split the audio output into <minutes> recorded chunks.
- -x: Write phoneme mnemonics to stdout.
- -X: Write phonemes mnemonics and translation trace to stdout. If rules files have been built with --compile=debug, line numbers will also be displayed.
- --path=<path>: Specifies the directory containing the espeak-ng-data directory
- --voices[=<language code>]: Lists the available voices. If =<language code> is present then only those voices which are suitable for that language are listed. If xx-yy language code is passed, then voices with yy of xx language variants are shown with higher priority than just xx.

5.5.3 SYNTHESIS METHOD

eSpeakNG can be used as text-to-speech translator in different ways, depending on which text-to-speech translation step user want to use.

Step 1 – Text to phoneme translation:

There are many languages (notably English) which don't have straightforward one-to-one rules between writing and pronunciation; therefore, the first step in text-to-speech generation has to be text-to-phoneme translation.

- Input text is translated into pronunciation phonemes (e.g. input text xerox is translated into zi@r0ks for pronunciation).
- Pronunciation phonemes are synthesized into sound e.g., zi@r0ks is voiced as zi@r0ks in monotone way

To add intonation for speech i.e. prosody data are necessary (e.g. stress of syllable, falling or rising pitch of basic frequency, pause, etc.) and other information, which allows to synthesize more human, non-monotonous speech. For example, in eSpeakNG format stressed syllable is added using apostrophe: z'i@r0ks which provides more natural speech. If eSpeakNG is used for generation of prosody data only, then prosody data can be used as input for MBROLA diphone voices.

Step 2 – Sound synthesis from prosody data:

eSpeakNG provides two different types of formant speech synthesis using its two different approaches. With its own eSpeakNG synthesizer and a Klatt synthesizer. This is explained below in detail:

- The eSpeakNG synthesizer creates voiced speech sounds such as vowels and sonorant consonants by additive synthesis adding together sine waves to make the total sound. Unvoiced consonants e.g. /s/ are made by playing recorded sounds, because they are rich in harmonics, which makes additive synthesis less effective. Voiced consonants such as /z/ are made by mixing a synthesized voiced sound with a recorded sample of unvoiced sound.
- The Klatt synthesizer mostly uses the same formant data as the eSpeakNG synthesizer. But, it also produces sounds by subtractive synthesis by starting with generated noise, which is rich in harmonics, and then applying digital filters and enveloping to filter out necessary frequency spectrum and sound envelope for particular consonant (s, t, k) or sonorant (l, m, n) sound.

For the MBROLA voices, eSpeakNG converts the text to phonemes and associated pitch contours. It passes this to the MBROLA program using the PHO file

format, capturing the audio created in output by MBROLA. That audio is then handled by eSpeakNG.

5.6 APPLICATIONS

Speech synthesis has long been a vital assistive technology tool and its application in this area is significant and widespread. It allows environmental barriers to be removed for people with a wide range of disabilities. The longest application has been in the use of screen readers for people with visual impairment, but text-to-speech systems are now commonly used by people with dyslexia and other reading disabilities as well as by pre-literate children. They are also frequently employed to aid those with severe speech impairment usually through a dedicated voice output communication aid. Work to personalize a synthetic voice to better match a person's personality or historical voice is becoming available. A noted application, of speech synthesis, was the Kurzweil Reading Machine for the Blind which incorporated text-to-phonetics software based on work from Haskins Laboratories and a black-box synthesizer built by Votrax.

Fig 5.3: Computer and speech synthesizer housing used by Stephen Hawking in 1999

Speech synthesis techniques are also used in entertainment productions such as games and animations. In 2007, Animo Limited announced the development of a speech synthesis software based software application package FineSpeech, explicitly geared towards customers in the entertainment industries, able to generate narration and lines of dialogue. The application reached maturity in 2008, when NEC Biglobe announced a web service that allows users to create phrases from the voices of characters from the Japanese anime series Code Geass: Lelouch of the Rebellion R2. In recent years, text-to-speech for disability and impaired communication aids have become widely available. Text-to-

speech is also finding new applications; for example, speech synthesis combined with speech recognition allows for interaction with mobile devices via natural language processing interfaces. A speech synthesizer was used by Dr. Stephen Hawking, who had Motor Neuron Disease, to communicate with other people. Figure 5.3 shows this computer and speech synthesizer housing.

Text-to-speech is also used in second language acquisition. Voki, for instance, is an educational tool created by Oddcast that allows users to create their own talking avatar, using different accents. They can be emailed, embedded on websites or shared on social media. Tools, like Elai.io are allowing users to create video content with AI avatars who speak using text-to-speech technology. In addition, speech synthesis is a valuable computational aid for the analysis and assessment of speech disorders. A voice quality synthesizer, developed by Jorge C. Lucero et. al. at the University of Brasília, simulates the physics of phonation and includes models of vocal frequency jitter and tremor, airflow noise and laryngeal asymmetries. The synthesizer has been used to mimic the timbre of dysphonic speakers with controlled levels of roughness, breathiness and strain.

CHAPTER 6
HARDWARE SETUP

In Chapter 2 we learned about various Raspberry Pi boards and the devices that can be connected to Pi through various available ports such as HDMI, CSI, 3.5mm audio jack and GPIO pins. In this chapter, we will be discussing about the specific hardware components that we are going to use in our project, in detail.

6.1 HARDWARE COMPONENTS

6.1.1 RASPBERRY PI 3 MODEL B+

Raspberry Pi 3 Model B+ is the final revision of third-generation Raspberry Pi single-board computer. We have already seen about Raspberry Pi 3 in section 2.3 in brief but we are going to focus on just Raspberry Pi 3 Model B+ here, which is shown in the figure 6.1. It has a 64-bit quad core processor running at 1.4GHz, dual-band 2.4GHz and 5GHz wireless LAN, Bluetooth 4.2/BLE, faster Ethernet, and PoE capability via a separate PoE HAT The dual-band wireless LAN comes with modular compliance certification, allowing the board to be designed into end products with significantly reduced wireless LAN compliance testing, improving both cost and time to market. The Raspberry Pi 3 Model B+ maintains the same mechanical footprint as both the Raspberry Pi 2 Model B and the Raspberry Pi 3 Model B. A powerful feature of the Raspberry

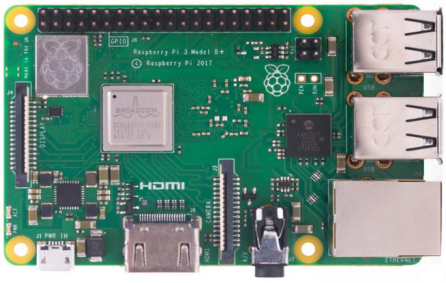


Figure 6.1: Raspberry Pi 3 Model B+

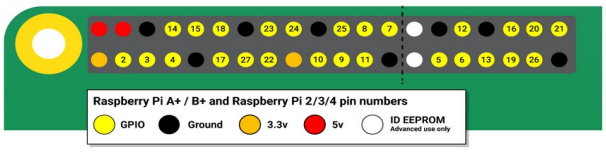


Figure 6.2: GPIO pins in Raspberry Pi 3

Pi 3 is the row of GPIO (general-purpose input/output) pins along the top edge of the board as shown in figure 5.2. A 40-pin GPIO header is found on all current Raspberry Pi boards which have a 0.1" (2.54mm) pin pitch. The pin configuration of a Raspberry Pi 3 40-pin GPIO is shown in figure 6.2. Two 5V pins and two 3.3V pins are present on the board, as well as a number of ground pins (0V), which are unconfigurable. The remaining pins are all general purpose 3.3V pins, meaning outputs are set to 3.3V and inputs are 3.3V-tolerant. A GPIO pin designated as an output pin can be set to high (3.3V) or low (0V).

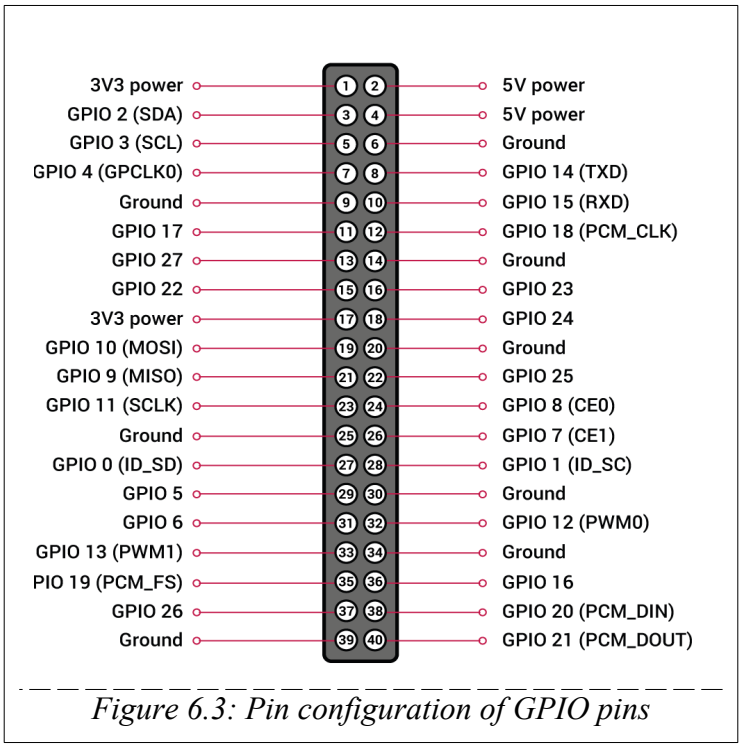


Figure 6.3: Pin configuration of GPIO pins

These pins can be used with a variety of alternative functions, some available on all pins, others on specific pins. These alternate functions are:

- PWM (Pulse-Width Modulation)
  - Software PWM available on all pins
  - Hardware PWM available on GPIO12, GPIO13, GPIO18, GPIO19
- SPI

- SPI0: MOSI (GPIO10); MISO (GPIO9); SCLK (GPIO11); CE0 (GPIO8), CE1 (GPIO7)
- SPI1: MOSI (GPIO20); MISO (GPIO19); SCLK (GPIO21); CE0 (GPIO18); CE1 (GPIO17); CE2 (GPIO16)

- I2C
  - Data: (GPIO2); Clock (GPIO3)
  - EEPROM Data: (GPIO0); EEPROM Clock (GPIO1)
- Serial
  - TX (GPIO14); RX (GPIO15)

The the detailed technical specifications of a Raspberry Pi 3 Model B+ is given in table 6.1.

Processor:	Broadcom BCM2837B0, Cortex-A53 64-bit SoC @ 1.4GHz
Memory:	1GB LPDDR2 SDRAM
Connectivity:	• 2.4GHz and 5GHz IEEE 802.11.b/g/n/ac wireless • LAN, Bluetooth 4.2, BLE • Gigabit Ethernet over USB 2.0 (maximum throughput 300Mbps) • 4 × USB 2.0 ports
Access:	Extended 40-pin GPIO header
Video & sound:	• 1 × full size HDMI • MIPI DSI display port • MIPI CSI camera port • 4 pole stereo output and composite video port
Multimedia:	H.264, MPEG-4 decode (1080p30); H.264 encode (1080p30); OpenGL ES 1.1, 2.0 graphics
SD card support:	Micro SD format for loading operating system and data storage

Input power:	• 5 V/2.5A DC via micro USB connector • 5V DC via GPIO header • Power over Ethernet (PoE)–enabled (requires separate PoE HAT)
Environment:	Operating temperature, 0–50°C
Production lifetime:	The board will remain in production until at least January 2023.

Table 6.1: Technical specifications of Raspberry Pi 3 Model B+

The physical specifications of the Raspberry Pi 3 Model B+ board is represented graphically in figure 6.4.

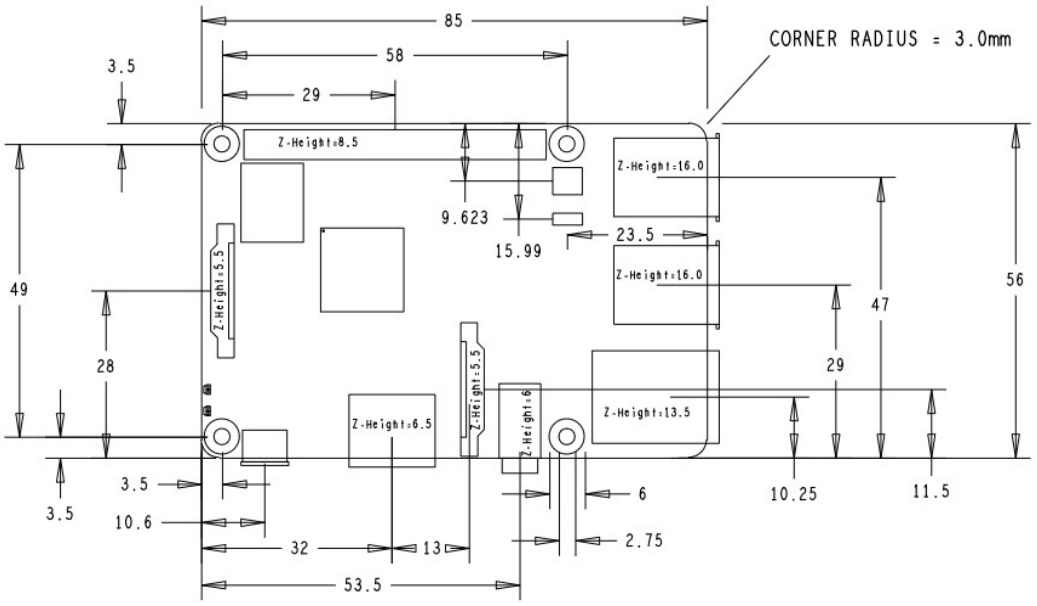


Figure 6.4: Physical specifications of Raspberry Pi 3 Model B+

6.1.2 RASPBERRY PI CAMERA MODULE

A standard Raspberry Pi Camera Module 3 is used in the project (shown in figure 6.5) which is a compact camera offering an IMX708 12-megapixel sensor with HDR, and features phase detection autofocus. Camera Module 3 is available in standard and wide-angle variants, both of which are available with or without an infrared cut filter. We are using a standard version of this camera module for this project. Camera Module 3 can be used to take full HD video as well as stills photographs, and features an HDR mode up to 3 megapixels. Its operation is fully supported by the libcamera library, including Camera Module 3's rapid autofocus feature offering plenty for both beginner and advanced users. Camera Module 3 is compatible with all Raspberry Pi boards with a Camera Serial Interface (CSI) Port. All variants of Camera Module 3 feature:

Figure 6.5: A standard Raspberry Pi Camera Module 3

- Back-illuminated and stacked CMOS 12-megapixel image sensor (Sony IMX708)
- High signal-to-noise ratio (SNR)
- Built-in 2D Dynamic Defect Pixel Correction (DPC)
- Phase Detection Autofocus (PDAF) for rapid autofocus
- QBC Re-mosaic function
- HDR mode (up to 3 megapixel output)
- CSI-2 serial data output
- 2-wire serial communication (supports I2C fast mode and fast-mode plus)
- 2-wire serial control of focus mechanism

The complete specifications of Raspberry Pi Camera Module 3 is given in table 6.2.

Sensor	Sony IMX708
Resolution	11.9 megapixels
Sensor size	7.4mm sensor diagonal

Pixel size	1.4µm × 1.4µm
Horizontal/vertical	4608 × 2592 pixels
Common video modes	1080p50, 720p100, 480p120
Output	RAW10
IR cut filter	Integrated in standard variants; not present in NoIR variants
Autofocus system	Phase Detection Autofocus
Dimensions	25 × 24 × 11.5mm (12.4mm height for Wide variants)
Ribbon cable length	200mm
Cable connector	15 × 1mm FPC
Operating temperature	0°C to 50°C
Compliance	FCC 47 CFR Part 15, Subpart B, Class B Digital Device Electromagnetic Compatibility Directive (EMC) 2014/30/EU Restriction of Hazardous Substances (RoHS) Directive 2011/65/EU
Production lifetime	Will remain in production until at least January 2030

Table 6.2: Technical specifications of Raspberry Pi Camera Module 3

The physical specifications of the Raspberry Pi Camera Module 3 is represented graphically in figure 6.6.

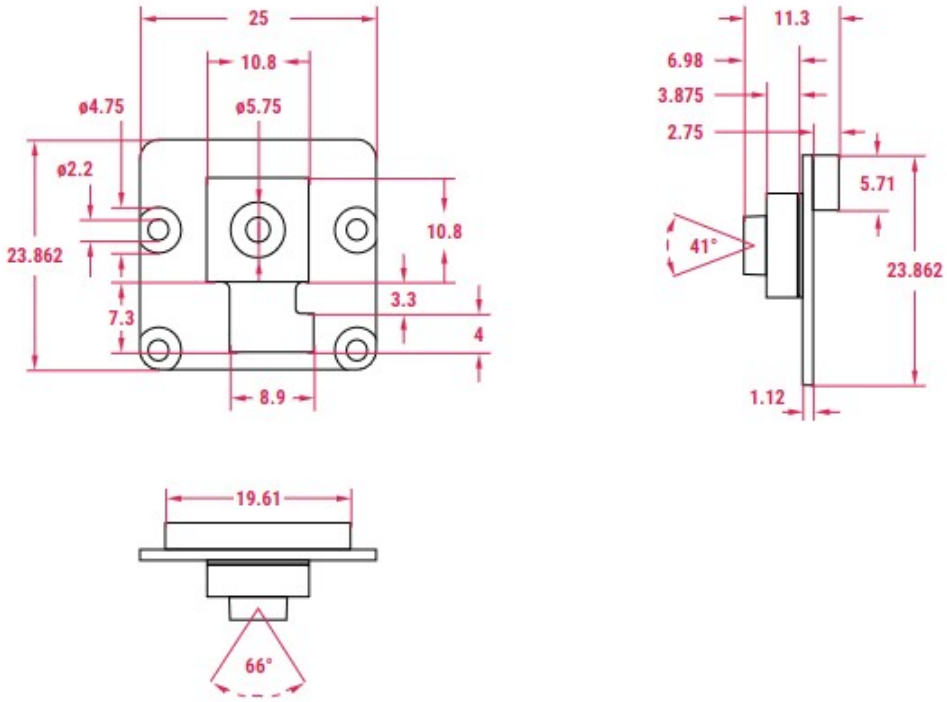


Figure 6.6: Physical specifications of a standard Raspberry Pi Camera Module 3

6.1.3 PUSH-BUTTON

A push button (shown in figure 6.7) switch is a mechanical device used to control an electrical circuit in which the operator manually presses a button to actuate an internal switching mechanism. Pressure is placed on the button or actuator, resulting in the depression of the internal spring and contacts and the touching of stable contacts at the bottom of the switch. This process will either close or open the electrical circuit. The push button switch generally consists of a button cap, a return spring, a bridge-type moving contact, a static

Figure 6.7: A push-button switch

contact, a pillar connecting rod and a shell. When the composite push button switch presses the button cap, the bridge-type moving contact moves downwards, the Normally Closed (NC) contact is opened first, and then the Normally Opened (NO) contact is closed. When the button cap is released, the NO contact is first broken and reset, and then the NC contact is closed and reset. This is the operation of a bush-button switch.

6.2 CONNECTING PI CAMERA TO RASPBERRY PI 3

The flex cable inserts into the connector labelled CAMERA on the Raspberry Pi, which is located between the Ethernet and HDMI ports. This port is called as the CSI (Camera Serial Interface) port. The cable must be inserted with the silver contacts facing the HDMI port. To open the connector, pull the tabs on the top of the connector upwards, then towards the Ethernet port. The flex cable should be inserted firmly into the connector, with care taken not to bend the flex at too acute an angle. To close the connector, push the top part of the connector towards the HDMI port and down, while holding the flex cable in place.

Once connected, the silver contacts must be uniformly inserted without any angular deviation of cable from the CSI connector. Figure 6.8 shows a properly connected camera module.

Figure 6.8: Camera module connected to Raspberry Pi 3

6.3 CONNECTING PUSH-BUTTON TO RASPBERRY PI GPIO

Connecting the Raspberry Pi's general purpose input output ports (GPIO) to a momentary tactile push button switch is a fairly simple circuit. We will need the following tools to complete the project:

- Raspberry Pi 3 Model B+
- Solderless breadboard
- 2 Jumper wires
- A 330Ω Resistor
- Push-button switch

We connect one side of the switch to an input pin on the Raspberry Pi, in this case we use pin 10. The other side of the switch we connect to 3.3V on pin 1 using a resistor. The resistor is used as a current limiting resistor to protect our input pin by limiting the amount of current that can flow. The circuit looks as shown in figure 6.9.

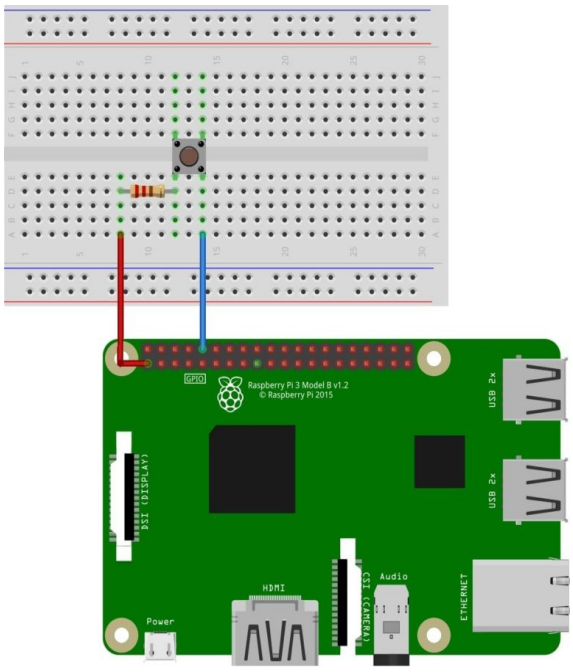


Figure 6.9: Push-button connection to Raspberry Pi 3

CHAPTER 7
SOFTWARE SETUP

7.1 OPERATING SYSTEM

An operating system is the set of basic programs and utilities that make the Raspberry Pi function properly. We are using Raspberry Pi OS (which was previously known as Raspbian) as the operating system. Raspberry Pi OS is a free operating system based on Debian optimized for the Raspberry Pi hardware. It is a Unix-like operating system based on the Debian Linux distribution for the Raspberry Pi family of compact single-board computers. First developed independently in 2012, it has been produced as the primary operating system for these boards since 2013, distributed by the Raspberry Pi Foundation. Raspberry Pi OS is highly optimized for the Raspberry Pi with ARM CPUs.

7.1.1 RASPBERRY PI OS VERSIONS

Raspberry Pi OS is produced in three installation versions:

- Raspberry Pi OS Lite (32-bit & 64-bit)
- Raspberry Pi OS with desktop (32-bit & 64-bit)
- Raspberry Pi OS with desktop and recommended software (32-bit)

There are also two legacy versions which are based on older versions of Debian:

- Raspberry Pi OS Lite (Legacy) (32-bit)
- Raspberry Pi OS (Legacy) with desktop (32-bit)

Raspberry Pi OS Lite is the smallest version, and does not include a desktop environment. The desktop version includes the Pixel desktop environment. Raspberry Pi OS with desktop and recommended software comes pre-installed with additional productivity software, such as Libre Office.

On December 2, 2021, the Raspberry Pi Foundation released Raspberry Pi OS (Legacy), a branch of the operating system that continued to receive security and hardware compatibility updates but was based on Buster, an older version of Debian.

All versions are distributed as disk image files, having the file extension img, intended to be flashed to microSD cards from which Raspberry Pi OS is booted. In March 2020, the Raspberry Pi Foundation published the Raspberry Pi Imager, a custom disk installer for Raspberry Pi OS, as well as other operating systems designed for the Raspberry Pi, including RetroPie, and Kodi OS,

The Raspberry Pi documentation recommends at least a 4 GiB microSD card for Raspberry Pi OS Lite, and at least an 8 GiB microSD card for all other versions.

7.1.2 PACKAGE MANAGER

Packages can be installed via APT and by using the Add/Remove Software tool, a GUI wrapper for APT. Advanced package tool, or APT, is a free-software user interface that works with core libraries to handle the installation and removal of software on Debian, and Debian-based Linux distributions. APT simplifies the process of managing software on Unix-like computer systems by automating the retrieval, configuration and installation of software packages, either from pre-compiled files or by compiling source code. APT is a collection of tools distributed in a package named apt. A significant part of APT is defined in a C++ library of functions; APT also includes command-line programs for dealing with packages, which use the library. Three such programs are `apt`, `apt-get` and `apt-cache`. They are commonly used in examples because they are simple and ubiquitous. The apt package is of "important" priority in all current Debian releases, and is therefore included in a default Debian installation.

APT can be considered a front-end to `dpkg`. While dpkg performs actions on individual packages, APT manages relations (especially dependencies) between them, as well as sourcing and management of higher-level versioning decisions (release tracking and version pinning). The user indicates one or more packages to be installed. Each package name is phrased as just the name portion of the package, not a fully qualified filename. For example, in a Debian system:

```
sudo apt-get install ack-grep
```

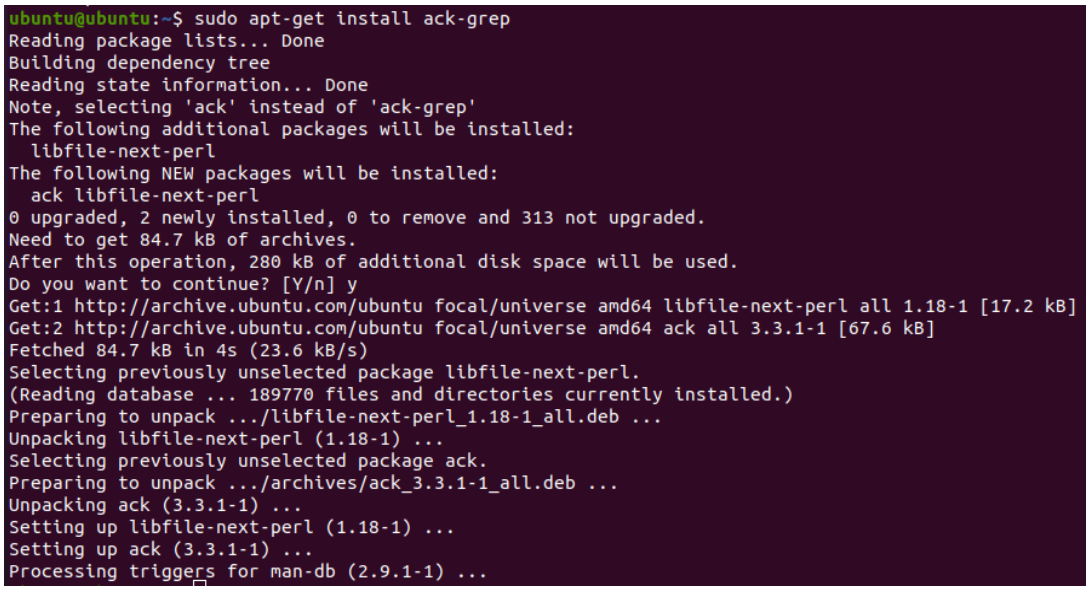


Figure 7.1: Example of 'apt-get install' command in Debian GNU/Linux based OS.

would be the argument used to install `ack_x.x.x-x_all.deb`. As shown in the figure 7.1, APT automatically gets and installs the requested package along with the packages, upon which the indicated package depends (if necessary). This was an original distinguishing characteristic of APT-based package management systems, as it avoided installation failure due to missing dependencies, a type of dependency hell.

To update the repository cache, package index and upgrade the distribution, we use certain apt and apt-get commands that facilitate the process such as:

- 'update' is used to resynchronize the package index files from their sources. The lists of available packages are fetched from the location(s) specified in `/etc/apt/sources.list`. For example, when using a Debian archive, this command retrieves and scans the Packages.gz files, so that information about new and updated packages is available.
- `upgrade` is used to install the newest versions of all packages currently installed on the system from the sources enumerated in `/etc/apt/sources.list`. Packages currently installed with new versions available are retrieved and upgraded; under no circumstances are currently installed packages removed, or packages not

already installed retrieved and installed. New versions of currently installed packages that cannot be upgraded without changing the install status of another package will be left at their current version.

- `full-upgrade` (apt) and `dist-upgrade` (apt-get), in addition to performing the function of upgrade, also intelligently handles changing dependencies with new versions of packages; apt and apt-get have a "smart" conflict resolution system, and will attempt to upgrade the most important packages at the expense of less important ones if necessary. The `/etc/apt/sources.list` file contains a list of locations from which to retrieve desired package files. Aptitude has a smarter dist-upgrade feature called full-upgrade.

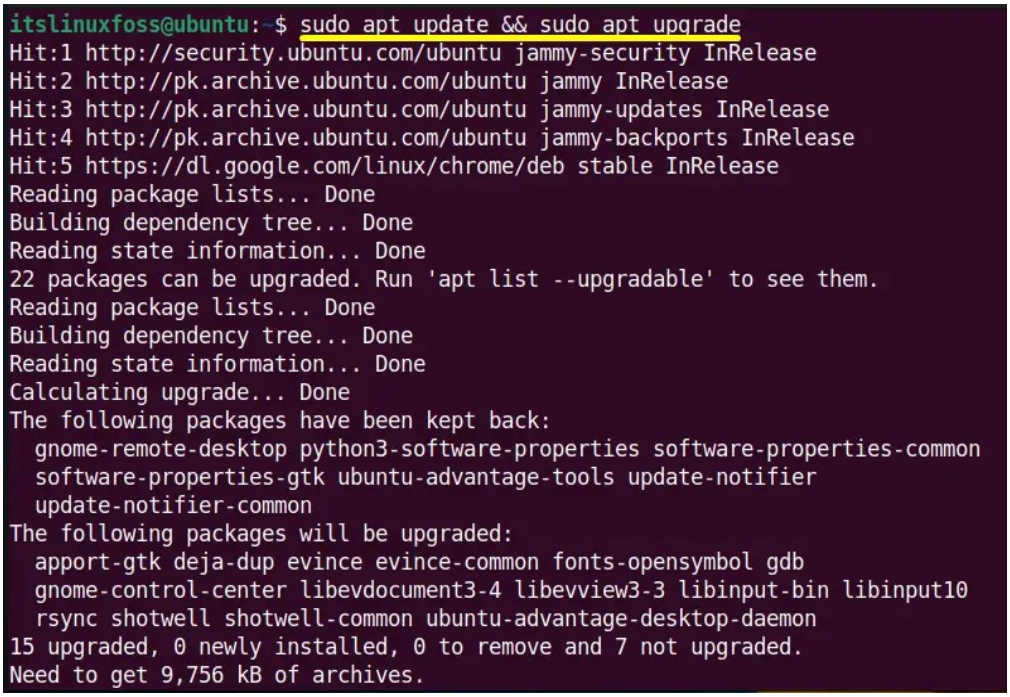


Figure 7.2: Upgrading installed packages using APT

As shown in figure 7.2, updating package index and repository cache and upgrading installed package can be done with a single command as shown below:

```
sudo apt-get update && sudo apt-get upgrade
```

7.1.3 DESKTOP ENVIRONMENT

Raspberry Pi OS has a desktop environment, PIXEL (short for Pi Improved Xwindows Environment, Lightweight) is shown in the figure 7.3. It is based on LXDE, which looks similar to many common desktops, such as macOS and Microsoft Windows. The desktop has a background image. A menu bar is positioned at the top and contains an application menu and shortcuts to a web browser (Chromium), file manager, and terminal. The other end of the menu bar shows a Bluetooth menu, Wi-Fi menu, volume control, and clock. The desktop can also be changed from its default appearance, such as repositioning the menu bar.

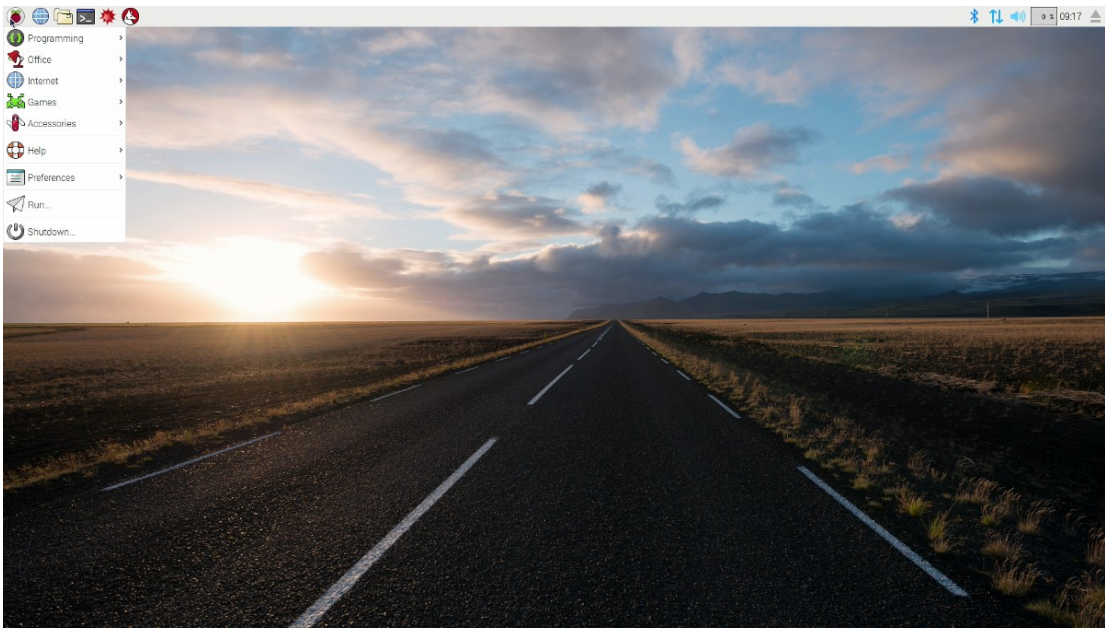


Figure 7.3: The PIXEL Desktop Environment

7.1.4 SOFTWARE COMPONENTS

Raspberry Pi OS uses PCManFM – a file browser allowing quick access to all areas of the computer, and was redesigned in the first Raspberry Pi OS Buster release (20th June, 2019). The web browser 'Epiphany' was originally shipped along with the OS, but switched to Chromium with the launch of its redesigned desktop. The built-in

browser comes pre-installed with uBlock Origin and h264ify. The image viewer pre-installed in the OS is `gpicview`.

Raspberry Pi OS comes with many beginner IDEs, such as Thonny Python IDE, Mu Editor, and Greenfoot. It also ships with educational software, such as Scratch and Bookshelf.

7.2 PYTHON SCRIPTING

We are using the Python programming language to script and automate the push-button operation, image capturing, capturing screenshot of the active window, image processing, text extraction and text-to speech conversion processes. Included into the Raspberry Pi operating system is the Thonny IDE for python development (shown in figure 7.4). An IDE is a nice word for a text editor that also allows us to execute and debug programs. The Python scripting done for this project will be discussed in-detail in this section.

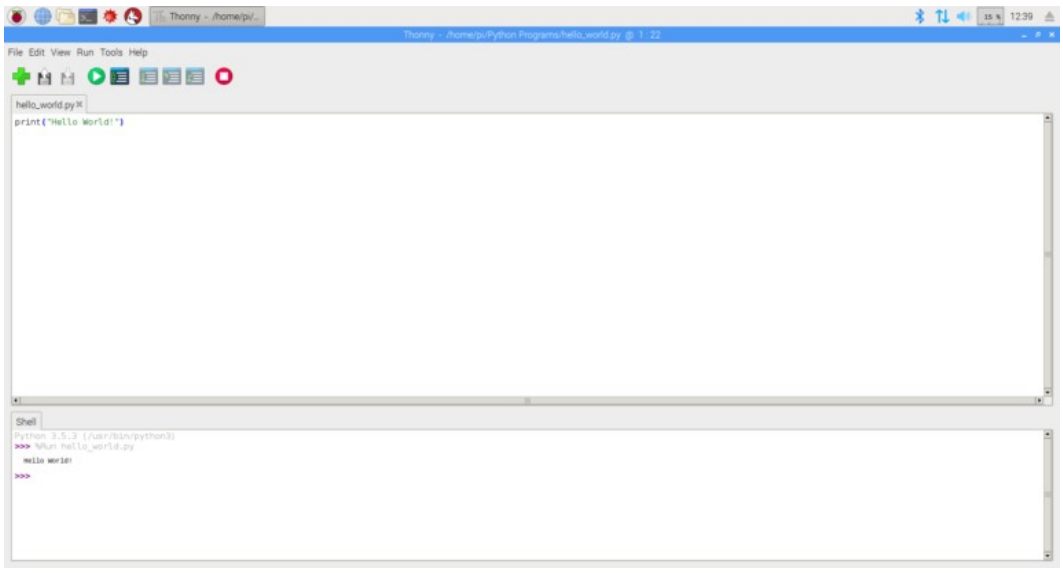


Figure 7.4: Thonny Python IDE

7.2.1 SENSING PUSH-BUTTON PRESSES

We have already discussed about the process of connecting a push-button to Raspberry Pi 3 Model B+. But we still need to actually poll the push-button to sense when it is pressed. This is achieved by Python scripting using `Rpi.GPIO` library, which is a Python module to control the GPIO on a Raspberry Pi. First, we install the python3-rpi.gpio package from the Debian repository:

```
sudo apt-get install python3-rpi.gpio
```

or by using Python Pip (Pip Installs Packages) to install packages to user virtual python environment:

```
pip install RPi.GPIO
```

After installing the python library, in the IDE, first we import the GPIO library and then setup the library to use board numbering. We then initialize the push-button pin connected to GPIO pin as an input pin and instruct the Raspberry Pi to use the in-built pull-down resister using the `pull_up_down` parameters. The GPIO initialization code looks as follows:

```
import RPi.GPIO as GPIO
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BOARD)
pin = # On-board GPIO pin number connected to push-button
GPIO.setup(pin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
```

We then use `GPIO.add_event_detect()` function to detect the button press event. This function accepts four arguments which are the GPIO pin number, `GPIO.RISING` or `GPIO.FALLING` – dictates if the event should be detected in the rising edge or falling edge of button press, `callback` – function which would be called when an event is detected and `bouncetime` – timeout for the polling function mentioned in milliseconds (ms). The argument parameters used in the function for our push-button is given below:

```
GPIO.add_event_detect(pin, GPIO.RISING, bouncetime = 500)
```

To count the number of push-button presses we use a count variable as counter.

The `GPIO.event_detected()` function outputs a Boolean value (True/False) which says the state of button – pressed or not pressed. This function along with `if...else` statement to increment count or break execution is looped unconditionally within a `While` loop. Python code for sensing the number of times the push-button is pressed (with the limit being 2 times):

```
count = 0
while True:
    event = GPIO.event_detected(pin)
    if event == True:
        count +=1
        print(count)
        sleep(1)
    if count == 2:
        break
    elif count == 1 and event == False:
        print(count)
        break
    GPIO.cleanup()
    print("Push-button was pressed", count, "time(s)")
```

7.2.2 PI CAMERA IMAGE CAPTURE

Newer Linux operating systems use the `libcamera` software library. Libcamera is an open source camera stack and framework for Linux, Android, and ChromeOS. Cameras are complex devices that need heavy hardware image processing operations. Control of the processing is based on advanced algorithms that must run on a programmable processor. This has traditionally been implemented in a dedicated MCU in the camera, but in embedded devices algorithms have been moved to the main CPU to save cost.

Blurring the boundary between camera devices and Linux often left the user with no other option than a vendor-specific closed-source solution.

To address this problem the Linux media community, collaborating with the industry developed a camera stack that is open-source-friendly while still protecting vendor core IP. Libcamera was born out of that collaboration and offers modern camera support to Linux-based systems, including traditional Linux distributions, ChromeOS and Android. Raspberry Pi Camera is controlled by libcamera in the latest version of the Raspberry Pi OS (based on Debian Bullseye), where RaspiCam which was used in previous versions of the OS. The Raspberry Pi OS switched to libcamera to drive the camera system directly from open source code running on ARM processors. The proprietary code running on the Broadcom GPU, and to which users have no access at all, is almost completely by-passed.

In Python programming we use the `Picamera2` Python library, which is the libcamera-based Python interface to make use of the Raspberry Pi Camera. Picamera2 directly uses the Python bindings supplied by libcamera, although the Picamera2 API provides access at a higher level. Picamera2 is the replacement for the legacy `PiCamera` Python library. It provides broadly the same facilities, although many of these capabilities are exposed differently. It provides a very direct and more accurate view of the Pi's camera system, and makes it easy for Python applications to make use of them. We need to first install Picamera2 Python library along with Qt5 and OpenGL Python bindings so that we can have a graphical preview of Pi Camera output. We can install these libraries using APT or Pip as shown below:

```
sudo apt install python3-picamera2 python3-pyqt5 python3-opengl
```

```
pip install picamera2 PyQt5 PyOpenGL
```

The first step is to import Picamera2 and show the camera preview. The preview window is implemented using the Qt GUI toolkit and uses GLES hardware graphics acceleration.

It is the most efficient way to display images on the screen when using X Windows. Python code for showing a QTGL preview is given below:

```
from picamera2 import Picamera2, Preview

from libcamera import controls

camera = Picamera2()

camera.start(show_preview=True)
```

The 12MP Raspberry Pi Camera Module 3 has autofocus feature which can be controlled through software. The autofocus (AF) state machine has 3 modes, and its activity in each of these modes can be monitored by reading the "AfState" metadata that is returned with each image. The 3 modes are:

- Manual - The lens will never move spontaneously, but the "LensPosition" control can be used to move the lens "manually". The units for this control are dioptres (1 / distance in metres), so that zero can be used to denote "infinity". The "LensPosition" can be monitored in the image metadata too, and will indicate when the lens has reached the requested location.
- Auto - In this mode the "AfTrigger" control can be used to start an autofocus cycle. The "AfState" metadata that is received with images can be inspected to determine when this finishes and whether it was successful, though we recommend the use of helper functions that save the user from having to implement this. In this mode too, the lens will never move spontaneously until it is "triggered" by the application.
- Continuous - The autofocus algorithm will run continuously, and refocus spontaneously when necessary.

The `set_controls()` function is used to control various parameter of the camera such as autofocus. This function is defined by `controls` library in `libcamera`, which

was already imported in the camera preview code. The continuous autofocus style is used to make camera refocus spontaneously, by using `AfMode` parameter. Also we need to run the autofocus algorithm quicker than the default autofocus speed, which is modified by using `AfSpeed` parameter. The full `set_controls()` function used is given below:

```
camera.set_controls({“AfMode”:controls.AfModeEnum.Continuous
, “AfSpeed”:controls.AfSpeedEnum.Fast})
```

The next step is to capture image and saving it at a particular location in the filesystem. This process can be done using `start_and_capture_file()` function, which takes location as an argument. After the image is captured, preview and camera needs to be stopped to save CPU cycles spent on passing through camera output to screen and unlocking camera to be used by another program, if needed. This is achieved by using `stop_preview()` and `stop()` functions. The image capturing and camera stop process is shown in the below Python code:

```
camera.start_and_capture_file(<path to store file>)
camera.stop_preview()
camera.stop()
```

7.2.3 ACTIVE WINDOW SCREENSHOT

Taking screenshot of a window is done using a Linux screenshotting program known as ‘Scrot’. Scrot is a minimalist Linux command line screen capturing application. It allows substantial degree of flexibility by specifying parameters on command line, including the ability to invoke a third-party utility to manipulate the resulting screenshot. Features of the program include the ability to limit the scope of capturing to a specific screen area, to set the delay (if needed to capture some menu or another UI element which is shown only when focused) and to specify the filename template using wildcards. Other features include creating thumbnails of the taken screenshots and specifying the quality of the

resulting image if lossy format is required.

To capture screenshot of a specific active window, `-u` is appended to the Scrot command. There is also an option to delay the process in seconds, which can be achieved by also appending `-d` to the command. The complete command looks like this:

```
scrot -d 5 -u <image_directory>/<image_name>.img
```

To run a Linux command from within Python we use the `subprocess.run()` function. To use the `run()` function, `run` library must be imported from `subprocess` module. Python's standard subprocess module provides most of the capabilities one would need to run external processes from Python but the subprocess.run extension was created to run command line processes in a polite way. The following Python program will wait for 5 seconds and take a screenshot of just the currently active window and store it in the location mentioned:

```
from subprocess import run

run(['scrot','-d','5','-u','<image_directory>/
<image_name>.img'], capture_output=True)
```

7.2.4 OPENCV IMAGE PROCESSING

We first need to install OpenCV in the operating system. For installing OpenCV along with its Python library in Raspberry Pi OS by using APT, the following command needs to be executed from the terminal emulator:

```
sudo apt install libopencv-dev python3-opencv
```

OpenCV Python module can also be installed using Pip using the following command:

```
pip install opencv-python
```

To use OpenCV in the Python program, we need to import the `cv2` module which is given by the `opencv-python` or `python3-opencv` package installed previously. The `cv2.imread()` function is used to load the image captured / screenshot captured from

the specified file. The image should be in the working directory or a full path of image should be given. The function also accepts a flag which specifies the way in which image should be read. The general syntax of this function is given below:

```
cv2.imread(<path_to_file>, <flag>)
```

The default flag value is `cv2.IMREAD_COLOR`. There are a total of three flags, which are:

- cv2.IMREAD_COLOR: It specifies to load a color image. Any transparency of image will be neglected. It is the default flag. Alternatively, we can pass integer value 1 for this flag.
- cv2.IMREAD_GRAYSCALE: It specifies to load an image in grayscale mode. Alternatively, we can pass integer value 0 for this flag.
- cv2.IMREAD_UNCHANGED: It specifies to load an image as such including alpha channel. Alternatively, we can pass integer value -1 for this flag.

The following code is used to load an image from the given location with the default flag and display the image on-screen in a window named “Image”:

```
import cv2

path = <location_of_image>

img = cv2.imread(path)

cv2.imshow(‘Image’, img)
```

Images captured by the Raspberry Pi Camera are in the BGR color space. If the Tesseract OCR gets an image object that is not RGB, it will try to convert it to RGB first. So its good practice to convert the captured BGR image to RGB first. The captured BGR camera image can be converted to RGB or even grayscale image which is simply one in which the only colors are shades of gray (monochromic shades) to make it easy for the

Optical Character Recognition software to extract text from image effectively with greater accuracy.

OpenCV ‘cv2.cvtColor()’ function is used to convert an image from one color space to another. There are more than 150 color-space conversion methods available in OpenCV.. To convert image from BGR to RGB we can use the `cv2.COLOR_BGR2RGB` parameter as shown in the following code:

```
rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

To convert captured BGR image directly into a grayscale image, we can use `cv2.COLOR_BGR2GRAY` parameter instead as shown below:

```
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
```

To remove noise and improve image clarity, `GaussianBlur()` and `divide()` functions. The usage of these functions is shown below:

```
blur = cv2.GaussianBlur(gray, (0,0), sigmaX=33, sigmaY=33)
```

```
divide = cv2.divide(gray, blur, scale=255)
```

7.2.5 TESSERACT TEXT EXTRACTION

Tesseract OCR is first installed in Raspberry Pi OS through APT using the following command:

```
sudo apt install tesseract-ocr tesseract-ocr-eng
```

In addition to the base `tesseract-ocr` package, the trained data for various languages such as English, Hindi, Tamil, Telugu etc., using their corresponding three letter identifier such as eng, hin, tam, tel etc. Tesseract OCR is now installed in the OS. But to make use of Tesseract OCR within a Python program, we need to install the Python wrapper named `pytesseract`. Python-tesseract can read all image types supported by the Pillow and Leptonica imaging libraries, including jpeg, png, gif, bmp, tiff and others. Additionally, if

used as a script, Python-tesseract also has the ability to not only write the output into a file, but also will print the recognized text. We now download pytesseract using Pip as shown below:

```
pip install pytesseract
```

The function used to extract text from image is `pytesseract.image_to_string()`. This function accepts two arguments. One argument is the variable holding path to image or path to image and the other is the configuration options, either given directly or stored in a variable. The various configuration options (as seen in section 4.5.2) are Page Segmentation Modes, OCR Engine Modes and adding user trained data files. A sample basic program for extracting texts from an image along with configuration flags and display the extracted texts on screen is shown below:

```
import pytesseract

import cv2

img = cv2.imread(<path_to_image>/<image_file_name>)

conf = r'--oem 3 --psm 6'

texts = pytesseract.image_to_string(img, config = conf)

print(texts)
```

The extracted text output can also be written to a text file (.txt) using Python's `open()` and `write()` functions. The `open()` function takes the following arguments:

```
open(<file>, mode='r', buffering=-1, encoding=None, errors=None, newline=None, closefd=True, opener=None)
```

The `file` argument is necessary and is a path-like object giving the path name (absolute or relative to the current working directory) of the file to be opened or an integer file descriptor of the file to be wrapped. `Mode` is an optional string that specifies the mode in which the file is opened. It defaults to `r` which means open for reading in text mode.

Other common values are `w` for writing (truncating the file if it already exists), `x` for exclusive creation, and `a` for appending (which on some Unix systems, means that all writes append to the end of the file regardless of the current seek position). We will be using mode `w` in this case to create a new file and write into it or write into the file if already exists. The code to write Tesseract output into a file is given below:

```
with open(f'{path}/output.txt','w') as f:
    f.write(texts)
    f.write('\n')
```

7.2.6 TEXT-TO-SPEECH

First, we install the `espeak-ng` - base package and the `espeak-ng-data` - data package containing synthesized speech based for different languages to the OS using APT. The following command needs to be executed in a terminal emulator:

```
sudo apt install espeak-ng espeak-ng-data
```

Then, we need to install the Python wrapper for espeak-ng installed in OS. This is done through Pip by using the following command:

```
pip install espeakng
```

eSpeakNG has the ability to read an entire text file or a few lines entered as part of the command. The general syntax of espeak-ng command and its flags have been discussed in the section 5.5. To use eSpeakNG inside Python, we need to initialize the speaker using the `espeakng.Speaker()` function as shown below:

```
import espeakng

speaker = espeakng.Speaker()
```

The various eSpeakNG parameters such as rate, pitch, Words Per Minute (WPM) and voice can be configured by using the initialization variable along with the corresponding parameter names as shown below:

```
speaker.rate = 130

speaker.pitch = 150

speaker.wpm = 140

speaker.voice = ‘es’
```

The `voice` parameter is used to determine the language to be spoken by the synthesizer and is defaulted to `en`, which is English.

To read out a line or variable containing strings from within Python program, we use the `say()` function. It can take two arguments. The first argument is either a typed text within quotes or a string variable which holds text and the second is a Boolean `wait4prev` argument which if `True`, eSpeakNG to wait until the previous speech is complete. The `wait4prev` parameter is `False` by default. A simple code to make eSpeakNG speak a given line is given below:

```
speaker.say(“Hello World!”)
```

An example code to make eSpeakNG speak output text from Tesseract OCR along with the `wait4prev` parameter is shown below:

```
img = <path_to_image>

image = cv2.imread(img)

texts = pytesseract.image_to_string(img)

speaker.say(texts, wait4prev= True)
```

7.3 FLOW DIAGRAM

The flow of program execution occurs in the following sequence:

- We first have a unconditionally iterating `while` loop. The loop consists of the push-button user method that counts the number of times the push-button is pressed (upto two times) as already discussed in section 7.2.1. If the push-button is pressed once, the flow of execution branches to capturing image with the Raspberry Pi Camera as discussed in section 7.2.2. If the push-button is pressed twice, the flow of execution branches to taking screenshot of active window as already discussed in section 7.2.3.
- The captured image / screenshot is now processed by OpenCV by converting BGR image to RGB or grayscale and various other processing techniques like Gaussian blur, dividing, thresholding and inverting as seen in section 7.2.4.
- After the image is processed of OpenCV, the processed image is given as input to Tesseract OCR through `pytesseract` and text is extracted from the image and written into a file for future usage as already seen in the section 7.2.5.
- Finally, the texts given as output from Tesseract OCR are stored in a variable. eSpeakNG is used to speak the texts by using `espeakng` as discussed in section 7.2.6.
- The program execution is now re-iterated back to the push-button method since the `while` loop is always `True`.

The flow diagram of the reader for the visually impaired is shown in figure 7.5.

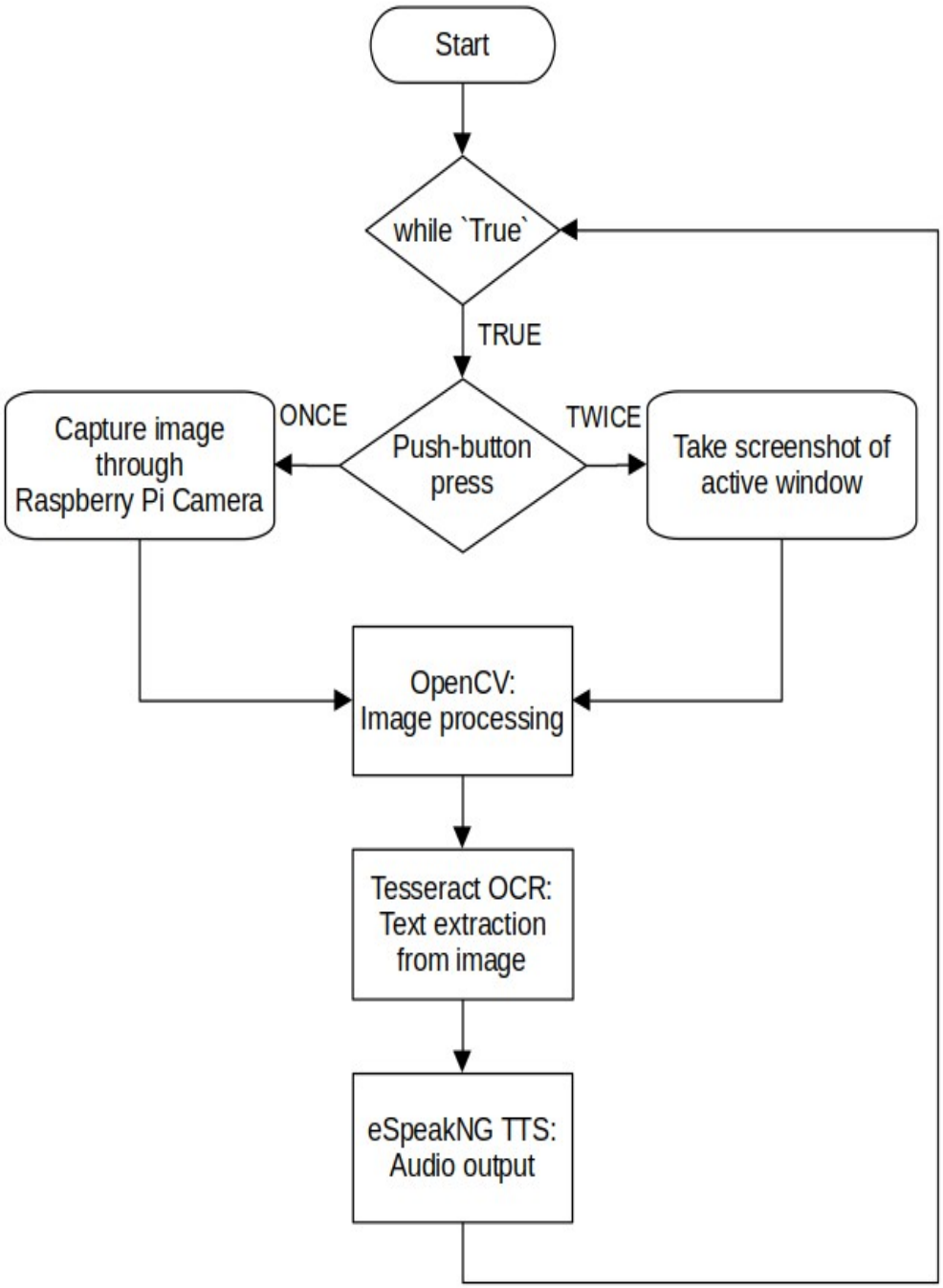


Figure 7.5: Flow diagram of Python program execution

CHAPTER 8
PROJECT RESULTS

8.1 IMAGE CAPTURED BY CAMERA

8.1.1 OUTPUT FROM CAMERA

As we have seen already in section 7.2.1, when we press the push-button once, the Raspberry Pi camera is activated and a BGR picture is taken as shown in figure 8.1. This is a picture of text written in a white A4 sheet paper with some watermarks to test the accuracy of image to text conversion process. The captured image is also stored in the path, `/home/pi/Project/Images/` as `captured.png`. This image, if given directly without any processing to Tesseract OCR, would not give us the desired text output. So the next step is OpenCV processing of this image. The content of the paper shown in figure 8.1 is:

"This is the DEMO of the Reader for Visually
Impaired – using Optical Character
Recognition, Open Source Computer Vision
and Raspberry Pi 3 Model B+.

This will recognize words:

HELLO WORLD!!

And numbers:

0123456789

from image captured by Camera"

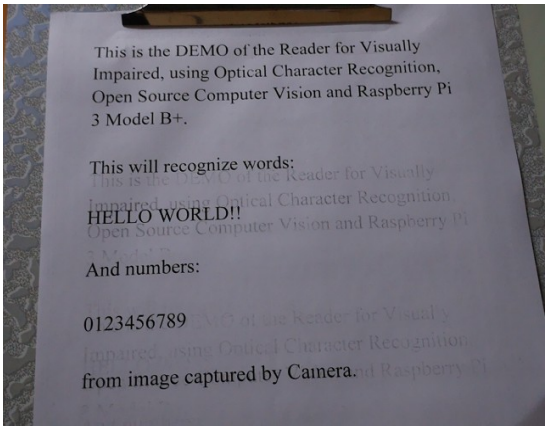


Figure 8.1: Image capture by Raspverry Pi Camera

8.1.2 OPENCV IMAGE PROCESSING

After the image is captured by camera, the BGR image is converted into RGB or grayscale as per requirement and various processing techniques are applied such as de-noising, thresholding, morphing, inverting are done as seen in section 7.2.4. Figure 8.2 is the BGR image converted into grayscale monochromatic image and stored in project images path as `opencv-gray.png`. The final processed, de-noised image is obtained after performing OpenCV `GaussianBlur()` and `divide()` functions and stored in the same path as `opencv-proc.png`. Figure 8.3 shows the final processed, de-noised image.

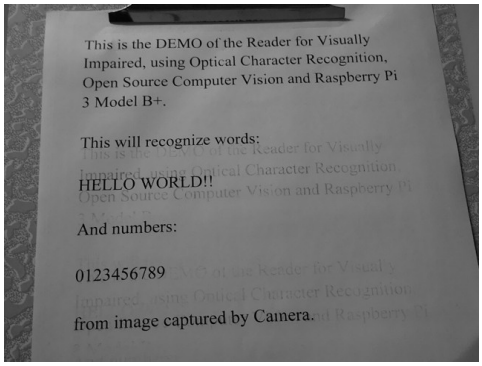


Figure 8.2: Captured image converted to grayscale using OpenCV

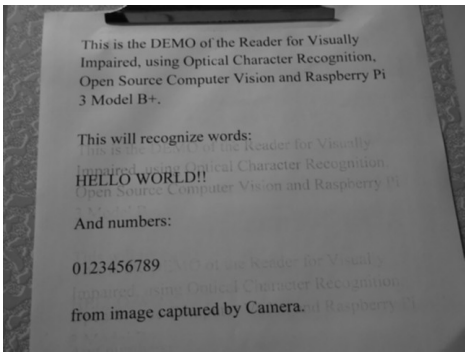


Figure 8.3: Processed de-noised version of grayscale image

8.1.3 EXTRACTED TEXT AND TTS OUTPUT

The final processed image is given as input to Tesseract OCR. The text is extracted from a camera image is upto 80% accurate which means that if there are 100 words, close to 80 words are exactly extracted without any deviation. Images from camera can have all kinds of imperfections causing the percentage of accuracy to go down. As the processed image is corrected for noise and unneeded grain in image, the OCR is now more accurately able to extract text from the image and this percentage goes higher. As we have seen in section 7.2.5, the extracted text is also written to a file named ‘output.txt’.

The contents of `output.txt` is shown in figure 8.4, which is:

“ees

This is the DEMO of the Reader for Visually
Impaired, using Optical Character Recognition,
Open Source Computer Vision and Raspberry Pi

3 Model B+.

This will recognize words:

HELLO WORLD!!
And numbers:

0123456789

from image captured by Camera.”

So, due to the image processing done by OpenCV, the extraction process is almost 95% accurate in this particular case. There are only a few mistakes made during text extraction by Tesseract OCR, which are:

- The ‘ees’ line is an erroneous detection due to the watermarks in paper in the image.
- The spacing and new line is not set exactly as seen in the image, which causes the TTS to improperly punctuate the sentences.

After extraction of texts, the text input is given to eSpeakNG module as discussed in section 7.2.6. The text is now converted into audio and is spoken by a robotic voice which is very clear, pleasant and understandable with a moderate words per minute and

pitch of voice.

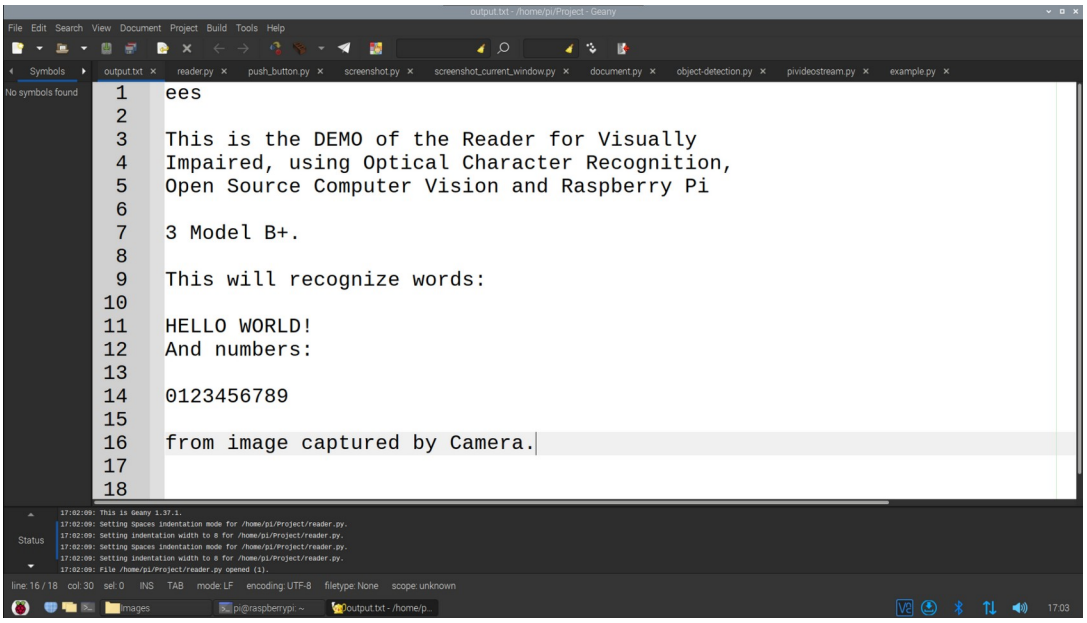


Figure 8.4: Tesseract output in `output.txt` file

8.2 SCREENSHOT OF ACTIVE WINDOW

8.2.1 SCROT IMAGE OUTPUT

As we have already discussed in section 7.2.3, Scrot is used to take screenshot of the currently active window. When the push-button is pressed twice, screenshot is taken instead of capturing an image using Pi Camera. Raspberry Pi OS's PIXEL desktop environment has a floating window manager, which means, multiple windows can be present on the screen at once and can overlap each other. Figure 8.5 shows that terminal emulator with our Python program running in the background and the document image is being viewed (active window) using `gpicview` - the image viewer. Scrot is set to wait for 5 seconds using `-d` flag and take screenshot of only the active window excluding the title bar of the application. The screenshot is shown in figure 8.6.

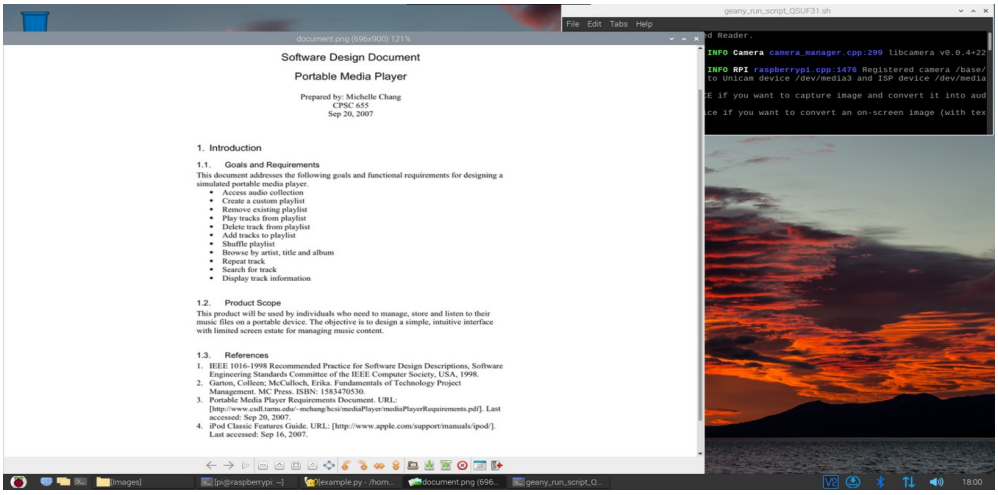


Figure 8.5: View of full desktop during screenshot process

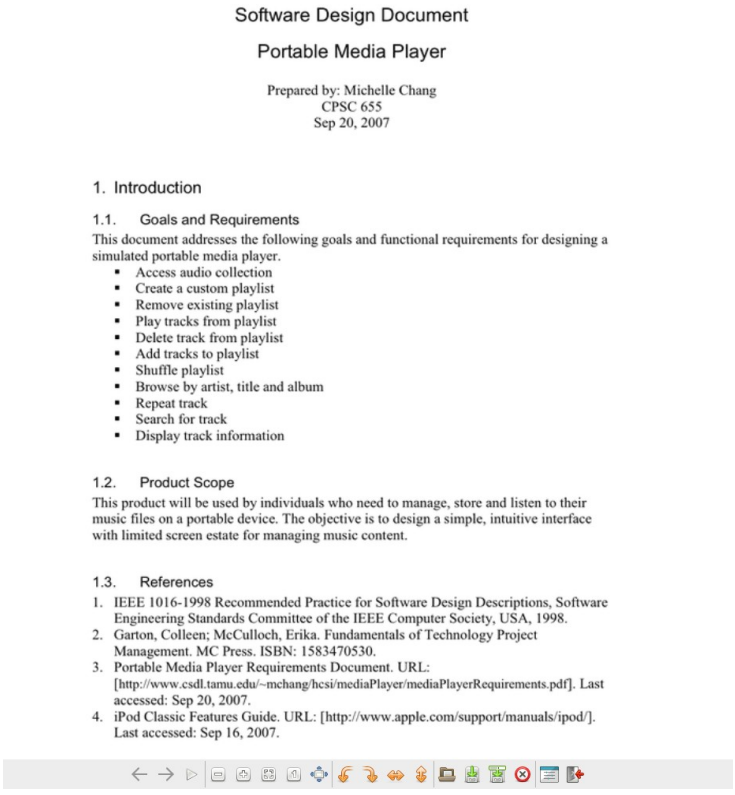
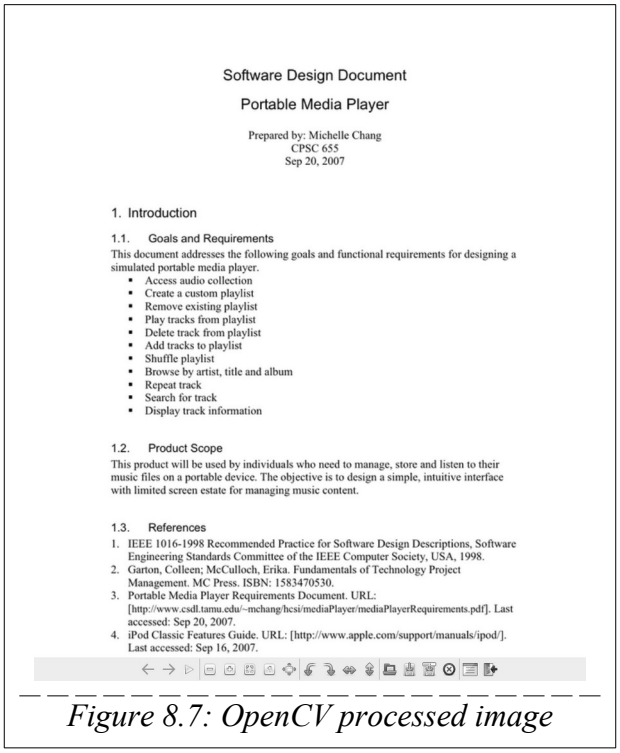


Figure 8.6: Screenshot of image viewer window, taken by Scrot

8.2.2 OPENCV IMAGE PROCESSING

Screenshots don't require the same level of processing as required for image captured by camera. This is because the text in screenshots are not subjected to any external noise occurring due to lack of steadiness or the amount of lighting. So, we only perform a transform to grayscale monochrome format from RGB as shown in figure 8.7.


Software Design Document

Portable Media Player

Prepared by: Michelle Chang
CPSC 655
Sep 20, 2007

1. Introduction

1.1. Goals and Requirements

This document addresses the following goals and functional requirements for designing a simulated portable media player.

- Access audio collection
- Create a custom playlist
- Remove existing playlist
- Play tracks from playlist
- Delete track from playlist
- Add tracks to playlist
- Shuffle playlist
- Browse by artist, title and album
- Repeat track
- Search for track
- Display track information

1.2. Product Scope

This product will be used by individuals who need to manage, store and listen to their music files on a portable device. The objective is to design a simple, intuitive interface with limited screen estate for managing music content.

1.3. References

1. IEEE 1016-1998 Recommended Practice for Software Design Descriptions, Software Engineering Standards Committee of the IEEE Computer Society, USA, 1998.
2. Garton, Colleen; McCulloch, Erika. Fundamentals of Technology Project Management. MC Press. ISBN: 1583470530.
3. Portable Media Player Requirements Document. URL: [http://www.csdl.tamu.edu/~mchang/hcsi/mediaPlayer/mediaPlayerRequirements.pdf]. Last accessed: Sep 20, 2007.
4. iPod Classic Features Guide. URL: [http://www.apple.com/support/manuals/ipod/]. Last accessed: Sep 16, 2007.

Figure 8.7: OpenCV processed image

8.2.3 EXTRACTED TEXT AND TTS OUTPUT

Tesseract text extraction in this case, is better and has a better accuracy of close to 90%. Issues during the extraction process can only be caused by different text colors, icons and glyphs shown on-screen such as a 'next', 'previous' button. Issues due to text coloring is rectified by converting the image to grayscale monochrome color space using OpenCV. The grayscale image is given as input to Tesseract OCR. Output from Tesseract is stored in `texts` variable as well as in `output_screenshot.txt` file as already discussed. The output stored in file is shown in figure 8.8.

Two mistakes were committed by Tesseract OCR during text extraction process, such as:

- The bullet points are incorrectly replaced by various mathematical symbols.
- All the bottom icons that are shown by `gpicview` image viewer (figure 8.7) are incorrectly read as 'oleeG0%\F os BAROEG' in the last line.

The text is now converted into audio by eSpeakNG in a pleasant and clearly comprehensible robotic voice.

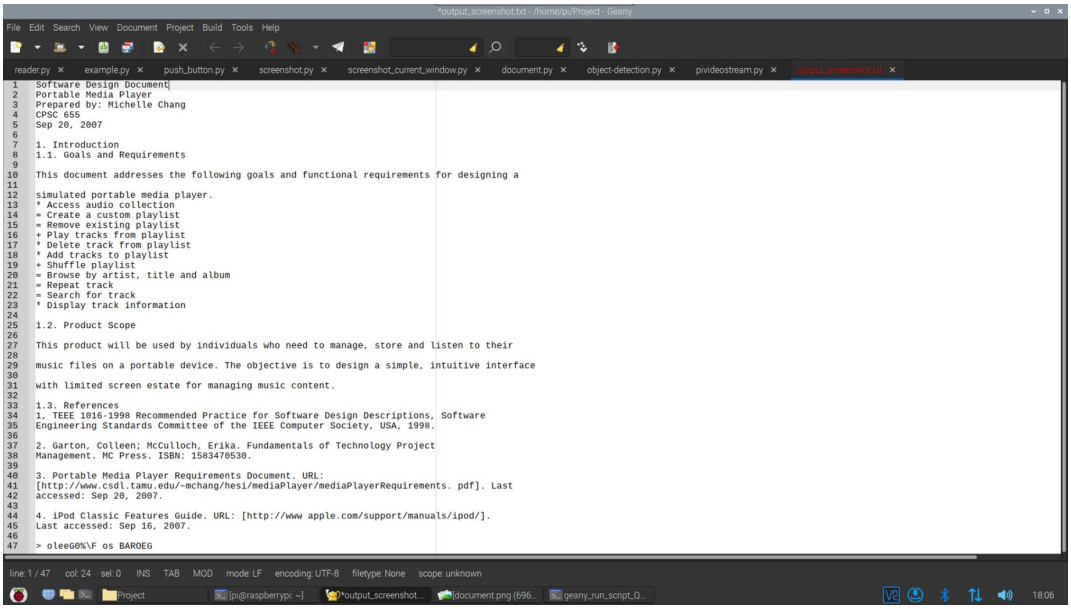


Figure 8.8: Text extracted from screenshot by Tesseract OCR

8.3 COST ESTIMATION & EXPENDITURE

A project budget is the estimated total costs projected to complete a project over a defined period of time. It is used to estimate what the costs of the project will be for every phase of the project. We Estimated Approximately ₹10000/- as budget for our project. After implementing the project fully, the amount spent is shown in the table 8.1.

S.No	Component Name	Quantity (Pcs)	Price (₹)
1.	Raspberry Pi 3 Model B+	1	3880
2.	Case for Raspberry Pi 3 Model B+	1	800
3.	Raspberry Pi Camera Module 3	1	2726
4.	Push-button switch	1	49
5.	Resistor - 330Ω	1	35
6.	Jumper wires (Male-Male)	2	84
7.	Jumper wires (Male-Female)	2	
8.	Mini-bread board	1	72
9.	Spectacle	1	450
10.	Plastic fixture	1	50
Total			8146

Table 8.1: Total expenditure

CHAPTER 9

9.1 CONCLUSION

The project is successful in implementing an easy-to-use reader which reads out the texts. The reader is able to take a picture through camera or take a screenshot of active application window and convert the texts in the images using OpenCV image processing, Tesseract OCR text extraction and eSpeakNG TTS audio output. All these processes are done with just a simple push-button press, which is very easy-to-use and intuitive for the blind and visually impaired people.

From our testing, we find that the reader is most efficient when presented with a plain white paper with black texts in foreground written in a popular font such as Times New Roman, Arial, Google Sans or Liberation Sans/Serif. The reader is 95% accurate in this case when the image is captured by camera i.e., the reader is able to read-out 9.5 out of 10 words exactly as it is written in paper. The reader's accuracy is further improved to 98% when the paper's image is taken from screenshot of image viewer.

When there are some imperfections in captured image or non-normal elements such as heavily stylized fonts, thick watermarks in background, colorful texts or multitude of UI icons (in case of a screenshot), the reader's accuracy falls to 80% – 85%, which is still good but not as good as the previous case. There is still scope for improvement in this area and accuracy can be improved.

9.2 FUTURE SCOPE

- Miniaturize the design in such as way that the entire board fits in the side frame of a spectacle.
- Making the OS lightweight by having only the essential packages and libraries installed so that the amount of internal storage required can be minimized. This directly contributes to on-board space savings of internal storage chip.

- Enhance the image processing and OCR algorithm such that any font in a captured image can be detected and converted into audio without any errors in the process.
- In captured images, currently only images of texts with white background can be accurately converted to speech.
- The screenshot functionality can be improved to accurately capture the active window without capturing its menu bar, tab bar and title bar.
- The internals such as the Processor and RAM can be upgraded to provide faster text extraction and audio conversion. This can be achieved by designing a custom board built from ground-up for this specific purpose. This would also reduce cost.

BIBLIOGRAPHY

[1] World Health Organization's (WHO) world report on vision (2019) - https://www.who.int/publications/i/item/9789241516570

[2] S. Thakare, A. Kamble, V. Thengne and U. R. Kamble, "Document Segmentation and Language Translation Using Tesseract-OCR", 2018 IEEE 13th International Conference on Industrial and Information Systems (ICIIS), Rupnagar, India, 2018, pp. 148-151, DOI: 10.1109/ICIINFS.2018.8721372 - https://ieeexplore.ieee.org/document/8721372

[3] Dr. Suraya Mubeen, Jally Brahmani, Datha Pavan Kalyan, Ayesha Jagirdar and A. Praveen Kumar, "Optical Character Recognition Using Tesseract", 2022 International Journal for Research in Applied Science and Engineering Technology. 10. 672-675. DOI:10.22214/ijraset.2022.47414 - https://www.ijraset.com/best-journal/optical-character-recognition-using-tesseract

[4] Himanshu Nath Tiwari, "Extract Text from Images in Python using OpenCV and EasyOCR", 2023 - https://www.researchgate.net/publication/369981694_Extract_Text_from_Images_in_Python_using_OpenCV_and_EasyOCR

[5] A. Nishajith, J. Nivedha, S. S. Nair and J. Mohammed Shaffi, "Smart Cap - Wearable Visual Guidance System for Blind," 2018 International Conference on Inventive Research in Computing Applications (ICIRCA), Coimbatore, India, 2018, pp. 275-278, doi: 10.1109/ICIRCA.2018.8597327 - https://ieeexplore.ieee.org/document/8597327

[6] Visionaid International ReadIt Air detailed specifications - http://www.visionaid.com/phpincludes/en/products/readit_air/readit_air_specs.php

[7] "Blind and Vision Impairment: Camera Scanners and OCR" recommended by the Australian Disability Clearinghouse on Education and Training (ADCET) - https://www.adcet.edu.au/inclusive-technology/blind-and-vision-impaired/camera-

scanners-ocr-and-text-to-speech

[9] HumanWare Australia Pty Limited listing of ReadIt Air - https://store.humanware.com/hau/readit-air.html

[10] The Eye-Pal Ace Plus users' manual - https://support.freedomscientific.com/Content/Documents/Manuals/Eye-Pal/Eye-Pal-Ace-Plus-User-Guide.pdf

[11] "Screen Readers" recommended by the American Foundation for the Blind (AFB) - https://www.afb.org/blindness-and-low-vision/using-technology/assistive-technology-products/screen-readers

[12] Hearing and Vision Center listing of Eye-Pal Ace Plus - https://hearingandvisioncenter.com/eye-pal-ace-plus/

[13] Raspberry Pi 3 Model B+ official website - https://www.raspberrypi.com/products/raspberry-pi-3-model-b-plus/

[14] Raspberry Pi Camera Module 3 official listing - https://www.raspberrypi.com/products/camera-module-3/

[15] OpenCV modules documentation - https://docs.opencv.org/4.x/

[16] Tesseract OCR User Manual - https://tesseract-ocr.github.io/tessdoc/

[17] eSpeakNG TTS Documentation in GitHub - https://github.com/espeak-ng/espeak-ng/blob/master/src/espeak-ng.1.ronn

APPENDIX

PICTURES OF THE PROJECT

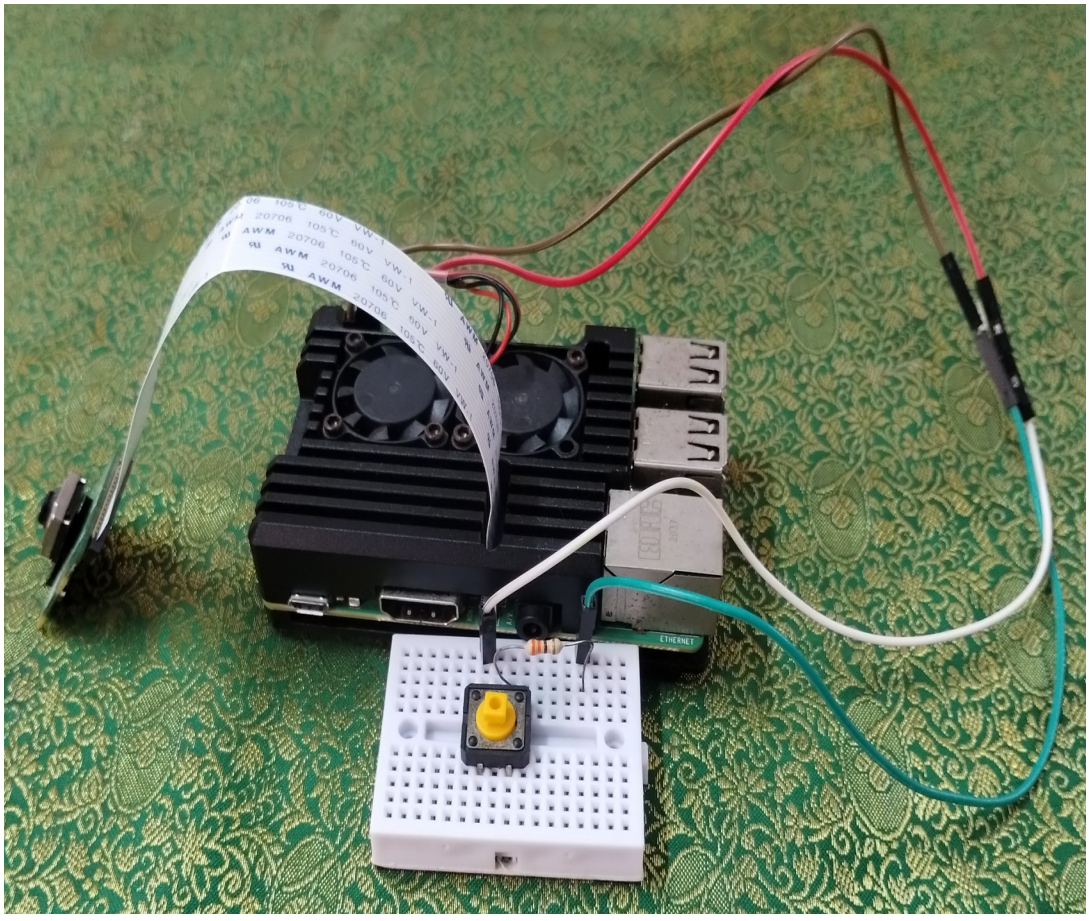


Figure A-1: Side View

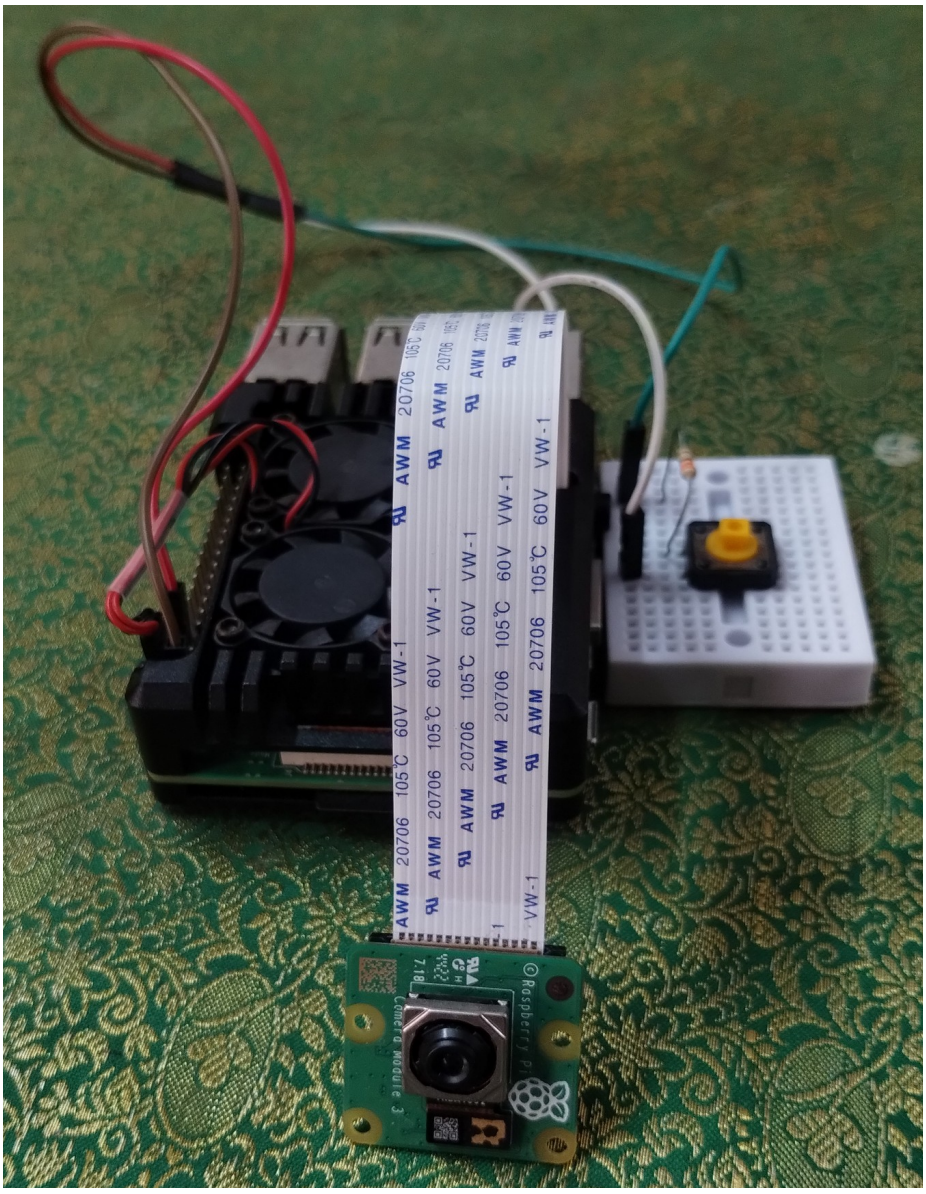


Figure A-2: Front View

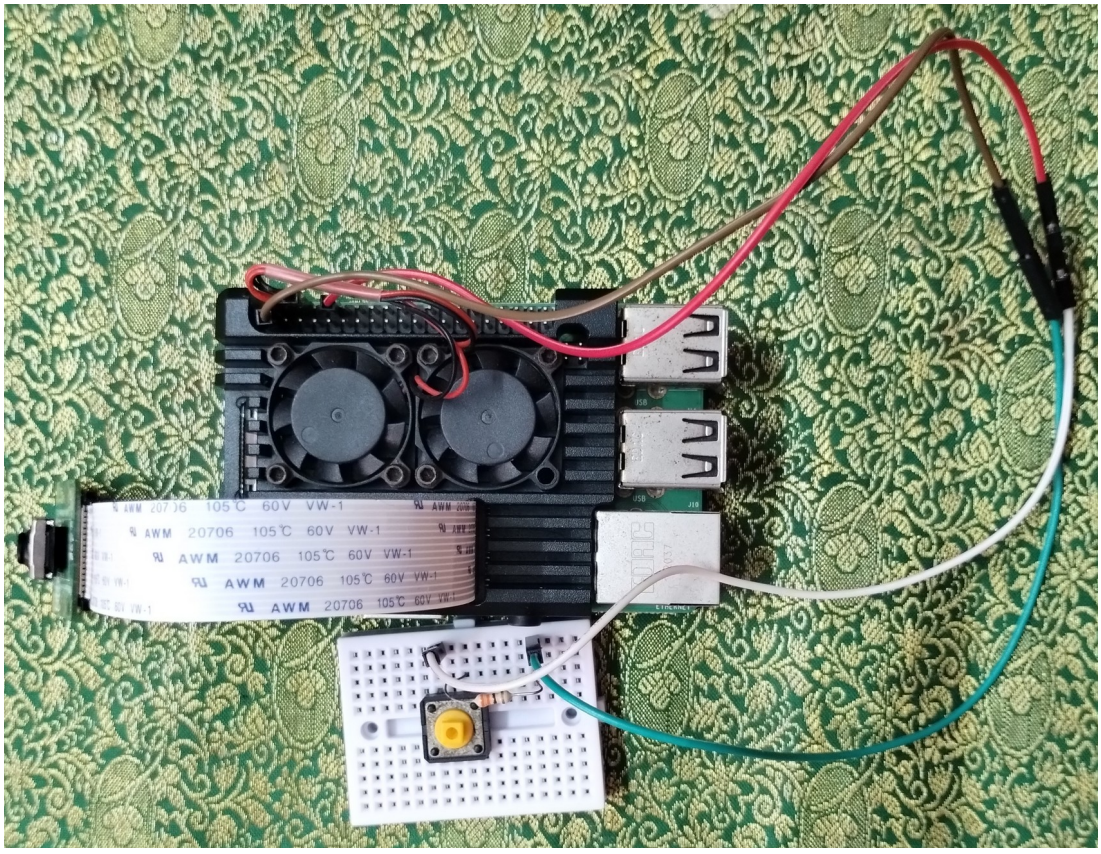


Figure A-3: Top View

SOFTWARE CODING

 # push_button.py – to detect the push-button presses.

```
def push_count():
  GPIO.setmode(GPIO.BOARD)
  pin = 10
  GPIO.setup(pin, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)
  GPIO.add_event_detect(pin, GPIO.RISING, bouncetime = 500)
  count = 0

  while True:
```

```
        event = GPIO.event_detected(pin)
        if event == True:
                count +=1
                print(count)
                sleep(0.3)
        if count == 2:
                break
        elif count == 1 and event == False:
                print(count)
                break
        GPIO.cleanup()

   print("Push-button was pressed", count, "time(s)")
   return count
```

 # screenshot_current_window.py – to get current active window's name

```
from subprocess import run

def get_focused_window():
  return run(['xdotool', 'getwindowfocus', 'getwindowpid', 'getwindowname'],
  capture_output=True).stdout.decode('utf-8').split()
```

 # screenshot.py – to take screenshot of active window (using scrot)

```
import cv2
import pytesseract
import espeakng
from subprocess import run
from screenshot_current_window import get_focused_window
```

```
def onscreen():
  speaker = espeakng.Speaker()
  speaker.rate=130

  # Get Focused window name with PID
  window = get_focused_window()
  speech = f“Taking screenshot of the {window} window”
  print(speech)
  speaker.say(“speech”, wait4prev = True)

  # Take screenshot
  sshot_loc = '/home/pi/Project/Images/screenshot.png'
  run([‘rm’, ‘/home/pi/Project/Images/screenshot.png’], capture_output=True)
  run([‘scrot’, ‘-d’, ‘5’, ‘-u’, ‘/home/pi/Project/Images/screenshot.png’],
capture_output=True)

  # OpenCV Reading image
  image = cv2.imread(sshot_loc)
  # Convert to GRAY
  img = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

  # Detect texts from image
  tess_config = r'--oem 1'
  texts = pytesseract.image_to_string(img, config=tess_config)
  print(texts)

  speaker.say(texts)
```

```
# reader.py – main Python program
import cv2
from PIL import Image
from picamera2 import Picamera2, Preview          # Raspberry Pi Camera and Preview
library
from picamera2.previews import QtGlPreview        # QtGlPreview from Camera
Previews
from libcamera import controls
from push_button import push_count
from screenshot import onscreen
from time import sleep                            # Sleep from Time library
import pytesseract                                # TesseractOCR library
import espeakng                                   # eSpeakNG library
# Initialize eSpeakNG TTS
speaker = espeakng.Speaker()
speaker.rate=130

print("Welcome to Raspberry Pi-based Reader.\n")
speaker.say("Welcome to Raspberry Pi-based Reader.")

# Initialize and start camera
camera = Picamera2()

# Push-button loop unconditionally
while True:
  print(">> Press the push-button ONCE if you want to capture image and convert it into
audio. <<\n>> Press the push-button twice if you want to convert an on-screen image
(with text) into          audio. <<\n")
```

```
 speaker.say("Press the push-button once if you want to capture image and convert it into audio.  Press the push-button twice if you want to convert an on-screen image (with text) into audio.",    wait4prev=True)
 btn_count = push_count()
 if btn_count == 1:
        # Capture image and stop camera preview and usage
        camera.start(show_preview=True)
        camera.set_controls({"AfMode": controls.AfModeEnum.Continuous, "AfSpeed":
                controls.AfSpeedEnum.Fast})
        sleep(5)

        camera.start_and_capture_file("/home/pi/Project/Images/captured.png")
        speaker.say("Image Captured. Please standby, saving image and processing...")
        camera.stop_preview()
        camera.stop()

        # OpenCV
        image = "/home/pi/Project/Images/captured.png"
        img = cv2.imread(image)
        # Convert to Grayscale and remove noise
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        blur = cv2.GaussianBlur(gray, (0,0), sigmaX=33, sigmaY=33)
        divide = cv2.divide(gray, blur, scale=255)

        # Detect texts from captured image
        #tess_config = r'--oem 1’
        texts = pytesseract.image_to_string(divide, config=tess_config)
        print(texts)
```

```
        # TTS Output from Tesseract output
        speaker.say(texts, wait4prev=True)
elif btn_count() == 2:
        onscreen()

sleep(0.5)
print("\nRepeating the process...\n")
speaker.say("Repeating the process...", wait4prev=True)
```