A Project Report on

VIRTUAL GAMING THROUGH HUMAN GESTURE DETECTION USING DEEP LEARNING

submitted in partial fulfillment of the requirement for the award of the Degree of

BACHELOR OF TECHNOLOGY

by

MOHAMMED ANAS AFFAF (18AT1A0513)
A S ABDUL KALAM (18AT1A0502)
MOULVI KAIF AHMED (18AT1A0557)
SHEIK ASLAM BASHA (18AT1A0522)

Under the Guidance of

M SRI LAKSHMI, M. Tech

Head of the Department

GPCET

Pioneering Innovative Education

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

G. PULLAIAH COLLEGE OF ENGINEERING AND TECHNOLOGY

(Accredited by NAAC of UGC with "A" Grade, Accredited by NBA (ECE, CSE & EEE)
Approved by AICTE, New Delhi, Recognized by UGC under 2 (f) & 12 (B) &
Permanently Affiliated to Jawaharlal Nehru Technological University Anantapur)
Pasupula (v), Nandikotkur Road, Kurnool – 518 452, Andhra Pradesh

2018-2022

G. PULLAIAH COLLEGE OF ENGINEERING AND TECHNOLOGY

(Accredited by NAAC of UGC with "A" Grade, Accredited by NBA (ECE, CSE & EEE)
Approved by AICTE, New Delhi, Recognized by UGC under 2 (f) & 12 (B) &
Permanently Affiliated to Jawaharlal Nehru Technological University Anantapur)
Pasupula (v), Nandikotkur Road, Kurnool – 518 452, Andhra Pradesh

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GPCET
Pioneering Innovative Education

CERTIFICATE

This is to certify that the project report entitled "**VIRTUAL GAMING THROUGH HUMAN GESTURE DETECTION USING DEEP LEARNING"** being submitted by **MOHAMMED ANAS AFFAF (18AT1A0513), A S ABDUL KALAM (18AT10A0502), MOULVI KAIF AHMED (18AT1A0557), SHEIK ASLAM BASHA (18AT1A0522)** in partial fulfillment of the requirement for the award of the degree of Bachelor of Technology in Computer Science and Engineering of Jawaharlal Nehru Technological University Anantapur, Ananthapuramu is a record of bonafide work carried out by them under my guidance and supervision.

The results embodied in this project report have not been submitted to any other university or institute for the award of any Degree or Diploma.

M. SRI LAKSHMI, M.Tech.
Project Supervisor

M. SRI LAKSHMI, M.Tech.,
Head of the Department

Date of Viva-Voce_____

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

We are extremely grateful to Chairman, **Sri G.V.M.Mohan Kumar,** of G.Pullaiah College of Engineering and Technology, Kurnool, Andhra Pradesh for their good blessings.

We owe indebtedness to our Principal **Dr.C.Srinivasa Rao,** M.E., Ph.D. G.Pullaiah College of Engineering and Technology, Kurnool for providing us the required facilities.

We would like to express our deep sense of gratitude and our sincere thanks to HOD **M Sri Lakshmi,** M.Tech.. Department of Computer Science and Engineering, G.Pullaiah College of Engineering and Technology, Kurnool for providing the necessary facilities and encouragement towards the project work.

We thank our project supervisor, **M Sri Lakshmi**, M.Tech., for her guidance, valuable suggestions and support in the completion of the project.

We gratefully acknowledge and express our thanks to teaching and non-teaching staff of CSE Department.

We would like to express our love and affection to our parents for their encouragement throughout this project work.

Project Associates

MOHAMMED ANAS AFFAF (18AT1A0513)
A S ABDUL KALAM (18AT1A0502)
MOULVI KAIF AHMED (18AT1A0557)
SHEIK ASLAM BASHA (18AT1A0522)

ABSTRACT

The Virtual environments are designed to provide natural, efficient, powerful, and flexible human-computer interaction. However, virtual worlds are not well-suited to the standard two-dimensional, keyboard-and-mouse-oriented graphical user interface. This will look at the most prevalent methods for capturing, monitoring, and recognizing many modalities at the same time in order to develop an intelligent human-computer interface for games. Given the wide range of gestures and their importance in building intuitive interfaces, the techniques under consideration concentrate on gestures, while they may be applied to other modalities as well. The methods under consideration are user-independent and do not need huge learning samples.

In this project, a model is developed that collects user gestures using a camera and uses the open-source free computer vision library OpenCV to convert real-time human motions into keyboard input for video game control.

Contents

LIST OF FIGURES AND TABLES

CHAPTER – 1
INTRODUCTION

CHAPTER 1

INTRODUCTION

1.1 HISTORY

We believe that virtual game development is an area in which we will enjoy working, but there is also a rapidly growing market in the gaming sector, despite the current economic downturn. If we look at the market share figures for gaming in the United States, we can see that the market share was worth USD$10.3 billion in 2004 and USD$65 billion in 2011.Some game development companies are based in Ireland, but they had the opportunity to meet with many more and hear about their current projects. Redwind, Havock, and Popcap are examples of such businesses. On November 4, 2010, the Kinect was released. The SDK's beta version was released on June 16, 2011, and the official version was released on February 1, 2012. Kinect holds the Guinness World Record for "Fastest Selling Consumer Electronic Device," with 8 million units sold in the first 60 days of its release. We could see the potential of Kinect when it first arrived on the market, and now, a year later, it has begun to fully exploit the technology's capabilities. Other technologies, such as the Wii or PlayStation Move, which both use motion technology, were considered. However, these technologies did not have as much functionality as our proposed method, which works similarly to the Kinect and allows you to track multiple elements without the use of a hand-held controller. Skeletal tracking, depth precision, voice control, and a standard camera are examples. It will have a significant impact on PCs without requiring the purchase of Kinect hardware.

1.2 NECESSITY

In today's world, people play motion sensing games on high-end, expensive consoles. These motion sensing games are made by companies like Microsoft, Sony, Nintendo, and Wii, and they are only compatible with consoles made by the same companies. Furthermore, these games are more costly than standard PC games. There are numerous advantages to PC gaming over console gaming. Consoles cannot be

upgraded to high graphics or processing power, whereas PCs can. The number of PC gamers worldwide is also significantly higher than that of console gamers. However, these PC games are played with a keyboard, mouse, and joystick. So, for this project, we concentrated solely on delivering a motion gaming experience on a PC. Through this attempt people who want to play PC games as they play motions sensing games, they don't need to buy costly consoles and Kinect hardware. All they need is to buy a standard camera and we will provide them a gesture set designed for specific input control.

1.3 OBJECTIVES

Objectives behind completion of this project are the following:

1. To improve gaming experience of PC gaming by enabling people to operate PC games through gestures.
2. To provide user friendly and easy to use gesture set for playing PC games.
3. To attract gamers toward PC gaming as there are lots of advantages of PC gaming over Console gaming.
4. To promote controller free gaming as this user will not need have games controller like keyboard, mouse or joysticks.
5. To offer experience of motion gaming for PC games without need of expensive gaming console and additional game controllers.
6. To deliver motion sensing gaming experience at very cheap cost as people will not need to buy consoles if they want to play normal PC games instead of expensive console games.
7. To investigate and research gesture technology using standard camera.
8. To investigate and research current uses of gesture recognition technology in current games.
9. To develop understanding of Python programming.

1.4 THEME

Following the above market trend, this project aims to improve the gaming experience by introducing a new way of working from an application standpoint, as there are no similar applications for Windows-based personal computers. The PC gamer will receive a gesture set tailored to the specific PC game. The main goal of the app, which sets it apart from other similar gaming apps, is the integration of motion detection and gaming application control, which eliminates the need to hold or even touch a keyboard, mouse, or controller device to play the game. Although the application models will have fixed types of moves and gestures, the actual gaming environment will be able to mimic your movements, similar to motion sensing games. However, you will gain experience with motion sensing games. This application may include a gesture set for a specific PC game. As a result, the user will only need a regular camera and not a gaming console. PC gaming has a number of advantages over console gaming. There are millions of PC games available on the market, and all of them can be played using gestures with this application.

1.5 PROBLEM DEFINATION

To implement cost effective system that read data from camera, detects human and accordingly perform action in order to

1. Play games through gestures (without using gaming consoles).
2. To control system functions through gestures.

CHAPTER – 2

LITERATURE SURVEY

CHAPTER 2

LITERATURE SURVEY

2.1 INTRODUCTION

Human detection is typically done in images captured with visible light cameras. Due to variations in pose, clothing, lighting conditions, and background complexity, detecting humans in images or videos is a difficult task. In recent years, there has been a lot of research into human detection, and various methods have been proposed. The majority of the research relies on images captured with visible-light cameras, which is a natural way of doing things similar to what human eyes do. Some methods use statistical training based on local features, such as gradient-based features like HOG and EOH, and others use image extraction techniques like scale-invariant feature transform (SIFT) and others. Because objects may not have consistent color and texture but must occupy an integrated region in space, depth information is an important cue when humans recognize them. In recent decades, there has been research into using range images for object recognition and modelling. Range images have several advantages over 2D intensity images: they are resistant to color and illumination changes. Range images are also simple representations of 3D data. However, due to lasers, earlier range sensors were expensive and difficult to use in human environments. Microsoft has now released the Kinect, which is both inexpensive and simple to use. It also lacks the drawbacks of lasers, allowing it to be used in a human environment and facilitating research in human detection, tracking and activity analysis. They presented a novel model-based method for human detection from depth images in the paper to which we referred. This method detects people's poses in indoor environments using standard camera data. It uses a pose detection process that includes a 2D edge detector that uses both the depth image's edge information. As a result, we've decided to use this method to detect human gestures and send commands to the back-end application, Games.

2.2 EXISTING SYSTEM

Today peoples are using high end gaming consoles and Kinect hardware for playing motion sensing games. The existing systems include popular gaming consoles like WII, Nintendo, PS1, PS2, PS3, and XBOX 360. Systems like WII or Play station Move which both use motion technologies. However, these technologies did not have as much functionality as our proposed method, which works similarly to the Kinect and allows you to track multiple elements without the use of a hand-held controller. These can include skeletal tracking, depth precision, voice control and a standard camera. These systems are based on the light sensing technology and require joysticks, light emitting devices etc.

Fig. 2.1 Existing Devices

2.3 LIMITATIONS OF EXISTING SYSTEM

- Console is required for playing games.
- The existing system can only be used for playing games.
- Light emitting sensors or joysticks are required in existing systems such as ps3 Nintendo, WII.
- Most of the games are compatible only with particular gaming console manufacturers.
- These motion sensing games are not compatible with PCs or laptops.

Fig. 2.2 Microsoft Kinect Sensor

CHAPTER – 3

SOFTWARE TOOLS

CHAPTER 3

SOFTWARE REQUIREMENTS SPECIFICATION

3.1 INTRODUCTION

Kinect is used with a console device for applications. We are attempting to solve the problem of skeletal tracking of a human body without using the Microsoft Kinect sensor because it is expensive, so we are attempting to optimize hardware by removing console devices and Kinect hardware.

3.1.1 PRODUCT SCOPE

Because of a number of complications such as variations in pose, lighting conditions, and the complexity of the background in the tracking environment, image processing-based human recognition remains a difficult task. This project presents a method for tracking human gestures using image processing techniques and pose data from a standard camera.

3.1.2 USER CLASSES AND CHARACTERISTICS

The users are most likely advanced gamers, children and people having hobby as gaming.

3.1.3 OPERATING ENVIRONMENT

- Windows 7 or 8 Onwards
- Intel x86/x64 architecture.
- The system is built on top of the Windows operating system. Windows 7 or 8 is required for the running python.

3.1.4 DESIGN AND IMPLEMENTATION CONSTRAINTS

The system performs its function using Python libraries like OpenCV. As far as the all that is required on his or her machine is a standard camera and the Python framework.

Software Requirements:

- Windows 7/8 onwards
- Anaconda
- Python Programming

Hardware Requirements:

- Pentium Processor IV or Higher
- Min 10 GB HDD
- RAM: 512MB or faster processor
- Standard Camera

3.1.5 ASSUMPTIONS AND DEPENDENCIES

- User has a large enough workspace area to accommodate the usage of a camera field of view.
- User has a computer system running Windows Operating System.
- The user has a powerful enough computer system that can handle the additional processing associated with OpenCV and frame media pipe introducing pose detection into the system.

3.2 SYSTEM FEATURES

1. System eliminates the console and allows us to play motion sensor games.
2. System detects human motion by using pose detection with maximum accuracy.
3. System allows us to play existing computer games using motion sensing technique.
4. System provides a user-friendly interface.

5. System provides us ability to operate Computer systems functionality through gestures.

3.3 EXTERNAL INTERFACE REQUIREMNTS

3.3.1 COMMUNICATIONS INTERFACES

- USB is used to communicate between camera and PC.
- Keyboard and mouse are used to emulate control.

3.4 NON-FUNCTIONAL REQUIREMENTS

3.4.1 PERFORMANCE REQUIREMENTS

1. The user should stand between 2 and 8 feet from camera.
2. The system should run in Real-time
    a. Should be able to track only 1 person in the Workspace.
    b. Should be able to scale up to detect human body pose in real time.
    c. Should be able to run on commercially available computing platforms
    d. Should be able to emulate input from mapped posed detected.

3.4.2 SAFETY REQUIREMENTS

The workspace should have no obstacles or obstructions in the way of the user. Elongated continuous usage of the system may be harmful, so it is highly recommended that two hours continued use sessions are followed by a fifteen-minute break with the system powered off.

A very small percentage of people may experience a seizure when exposed to certain visual images, including flashing lights or patterns that may appear in video games. Even people who have no history of seizures or epilepsy may have an undiagnosed condition that can cause these “photosensitive epileptic seizures” while observing the displays.

3.4.3 SOFTWARE QUALITY ATTRIBUTES

• Robustness - The system needs to be robust enough to generate a realistic virtual experience. The system should also be able to dynamically detect motions and body pose in real-time to generate accurate environment. The system needs to properly send system input based on detected pose.

• Reliability - The system should able to continuously run for a long duration of time (multiple hours) and not suffer from system slowdowns or crashes caused from memory leaks and zombie process.

• Portability - The software should be able to run on any Microsoft windows-based platform. To set up and tear down the entire system, displays need to be set in place. Camera need to be placed on the displays, Camera need to be connected to the computer, and the program need to be executed. Ideal set up/tear down time should be approximately two minutes.

• Ease of use - Someone with little or no technical experience in the operations of electronics should be able to setup and use this system by following a simple set of instructions.

• Ease of Learning - The learning curve for this software should be short since the software should perform the corresponding tasks based on natural human motions.

CHAPTER – 4

PROPOSED METHOD

.

CHAPTER 4

PROPOSED METHOD

4. PROPOSED METHOD

4.1 SYSTEM ARCHITECTURE

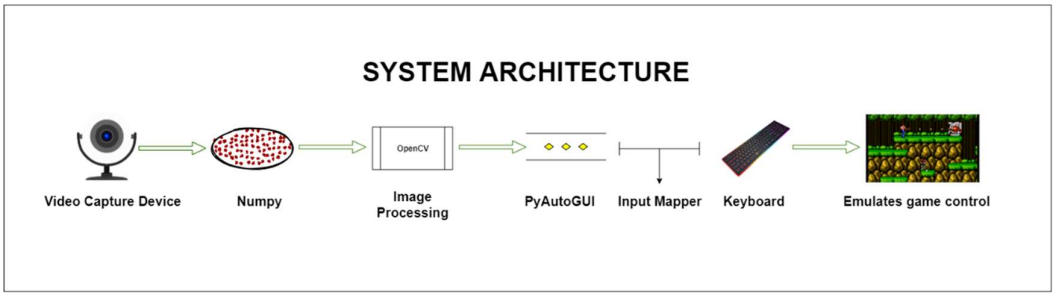

Fig. 4.1 Architecture

4.2 COMPONENTS OF ARCHITECTURE

4.2.1 VIDEO CAPTURE DEVICE

Video capture device used here is a standard webcam which either comes inbuilt or has to be connected externally via a USB cable. A webcam is a video camera that sends or streams an image or video to or through a computer network, such as the Internet, in real time. Small cameras that sit on a desk, attach to a user's monitor, or are built into the hardware are known as webcams. Webcams can be used during a video chat session between two or more people, with live audio and video conversations. Users can use webcam software to record or stream video over the Internet. Because video streaming over the Internet consumes a lot of bandwidth, compressed formats are commonly used. Because higher resolutions would be reduced during transmission, a webcam's maximum resolution is also lower than most handheld video cameras. Webcams are less expensive than most video cameras due to their lower resolution, but the effect is sufficient for video chat sessions. The term "webcam" (a clipped compound) can also refer to a video camera that is connected to the Internet indefinitely rather

than for a specific session, and that provides a view for anyone who visits its web page over the Internet. Some of them are expensive, rugged professional video cameras, such as those used as online traffic cameras. A webcam is a computer-connected camera. It can take still photos or video, and with the help of software, it can stream video over the Internet in real time. A webcam, such as the Logitech Webcam C270, is shown in the fig 4.2. Most webcams today are either built into the top edge of a laptop's display or connected to the computer's USB or FireWire port.

Fig. 4.2 Logitech c270 HD webcam

The XCoffee, also known as the Trojan Room coffee pot, was the first widely publicized webcam. With the help of Quentin Stafford-Fraser and Paul Jardetzky, the camera went live in 1991. In November 1993, Daniel Gordon and Martyn Johnson assisted in connecting the camera to the Internet. People didn't waste time going down to get coffee because the XCoffee webcam monitored a coffee pot outside the Trojan Room in the University of Cambridge. The website had over 150,000 people online watching the coffee pot after it was mentioned in the news. On August 22, 2001, the webcam was turned off. The image depicts how the XCoffee appeared on the internet.

4.2.2 NUMPY

NumPy is the most important Python package for scientific computing. It's a Python library that includes a multidimensional array object, derived objects

(such as masked arrays and matrices), and a variety of routines for performing fast array operations, such as mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation, and more. The Nd array object is at the heart of the NumPy package. Many operations are performed in compiled code for performance, and this encapsulates n-dimensional arrays of homogeneous data types. The following are some key distinctions between NumPy arrays and standard Python sequences:

- Unlike Python lists, NumPy arrays have a fixed size when they are created (which can grow dynamically). When you change the size of a Nd array, it creates a new one and deletes the old one.
- The elements of a NumPy array must all be of the same data type and thus have the same memory footprint. The exception is that arrays of (Python, including NumPy) objects can be used, allowing for arrays of various sizes.
- NumPy arrays facilitate advanced mathematical and other types of operations on large numbers of data. Typically, such operations are executed more efficiently and with less code than is possible using Python’s built-in sequences.
- NumPy arrays are used by a growing number of scientific and mathematical Python-based packages; while they typically support Python-sequence input, they convert it to NumPy arrays before processing, and they frequently output NumPy arrays. To put it another way, knowing how to use Python's built-in sequence types isn't enough to efficiently use much (if not all) of today's scientific/mathematical Python-based software; you also need to know how to use NumPy arrays.

It is the most important Python package for scientific computing. It is an open-source program. It has a number of features, including the following:

- A powerful N-dimensional array object
- Sophisticated (broadcasting) functions
- Tools for integrating C/C++ and Fortran code

- Useful linear algebra, Fourier transform, and random number capabilities

NumPy can be used as a multi-dimensional container of generic data in addition to its obvious scientific applications. NumPy allows arbitrary data types to be defined, allowing NumPy to integrate with a wide range of databases seamlessly and quickly. NumPy is a widely used general-purpose library that serves as the foundation for many other computation libraries, including SciPy, scikit-learn, TensorFlow, matplotlib, and OpenCV. Having a basic understanding of NumPy makes it easier to work with other higher-level libraries.

4.2.3 OPENCV

OpenCV (Open-Source Computer Vision Library) is an open-source library that contains hundreds of computer vision algorithms. Computer vision is a method for comprehending images and videos, as well as manipulating and retrieving information from them. Artificial Intelligence relies on or is primarily based on computer vision. Self-driving cars, robotics, and photo correction apps all use computer vision to some extent.

OpenCV is a large open-source library for computer vision, machine learning, and image processing, and it now plays an important role in real-time operations, which are critical in today's systems. It can be used to identify objects, faces, and even human handwriting in images and videos. Python can process the OpenCV array structure for analysis when it is combined with other libraries like NumPy. We use vector space and perform mathematical operations on these features to identify image patterns and their various features.

The first version of OpenCV was 1.0. OpenCV is free for both academic and commercial use because it is released under a BSD license. It supports Windows, Linux, Mac OS, iOS, and Android and has C++, C, Python, and Java interfaces. The main focus of OpenCV when it was created was real-time applications for computational efficiency. To take advantage of multi-core processing, everything is written in optimized C/C++.

Image processing is a technique for performing operations on images in order to improve them or extract useful information from them. If we talk about the basic definition of image processing then “Image processing is the analysis and manipulation of a digitized image, especially in order to improve its quality”. The goal of image processing is to assist a computer in comprehending an image's content. OpenCV is a programming library primarily used for image processing. It serves as the de facto API for computer vision applications. Image processing applications can help us solve a variety of real-time problems.

Image processing is a type of signal processing in which the input is an image, such as a photograph or video frame, and the output is an image or a set of image-related characteristics. OpenCV is a programming library primarily used for image processing. The open source Berkely Software Distribution license makes it freely available. Intel began the project as a research project. OpenCV includes a number of tools for dealing with computer vision issues. It includes low-level image processing functions as well as high-level face detection, feature matching, and tracking algorithms.

We are humans, and we can easily discern that the image is of a person who is me. However, if we ask the computer, "Is this my photo?" The computer is unable to respond because it is unable to solve the problem on its own. Any image is read by the computer as a range of values between 0 and 255. There are three primary channels in any color image: red, green, and blue.

OpenCV's main user interface is written in C++. Python, Java, and MATLAB now have complete user interfaces. Wrappers for other languages like C#, Perl, and Ruby have been created. Since 2010, a CUDA-based GPU interface has been in development. OpenCV has received support from Intel, as well as Willow Garage, a privately funded new robotic research institute. OpenCV runs on a variety of operating systems, including Windows, Android, Blackberry, OpenBSD, iOS, and Linux. The development of new OpenCV modules to support robotic perception is currently underway.

Applications of OpenCV: There are lots of applications which are solved using OpenCV, some of them are listed in next page

- face recognition
- Automated inspection and surveillance
- number of people – count (foot traffic in a mall, etc.)
- Vehicle counting on highways along with their speeds
- Interactive art installations
- Anomaly (defect) detection in the manufacturing process (the odd defective products)
- Street view image stitching
- Video/image search and retrieval
- Robot and driver-less car navigation and control
- object recognition
- Medical image analysis
- Movies – 3D structure from motion
- TV Channels advertisement recognition

4.2.4 PYAUTOGUI

PyAutoGUI allows your Python scripts to automate interactions with other applications by controlling the mouse and keyboard. The API is intended to be straightforward. PyAutoGUI runs on Python 2 and 3 and works on Windows, macOS, and Linux.

PyAutoGUI has several features:

- Moving the mouse and clicking in the windows of other applications.
- Sending keystrokes to applications (for example, to fill out forms).
- Take screenshots, and given an image (for example, of a button or checkbox), and find it on the screen.
- Locate an application's window, and move, resize, maximize, minimize, or close it (Windows-only, currently).

- Display alert and message boxes.

4.2.5 STANDARD INPUT DEVICE

A keyboard and mouse are used for the majority of computer interactions. The mouse allows the user to position the cursor, draw, and execute programme functions by clicking mouse buttons, while the keyboard allows the user to type letters and numbers. Many other actions can be performed on the keyboard using function keys, cursor keys, control keys, or keyboard macros in addition to basic typing. However, the mouse duplicates many of these actions. Many people nowadays prefer to use a mouse instead of keyboard keys to operate their computers. It's important to align the center of your body with the center of the keyboard part you use the most, which is usually either the alphabetic area or the numeric keypad, for good posture.

Fig. 4.3 Keyboard and Mouse

4.2.6 EMULATOR

A video game console emulator allows a computing device to emulate the hardware of a video game console and play its games on the emulating platform. Emulators frequently include extra features that go beyond the limitations of the original hardware, such as broader controller compatibility timescale control, increased performance, improved clarity, easier access to memory modifications (like Game Shark), one-click cheat codes, and unlocking of gameplay features.

Emulators are also useful in the creation of new games for older, discontinued, or rare consoles, as well as in the development of homebrew demos.

A ROM file (a copy of game cartridge data) or an ISO image (a copy of optical media) is typically used to supply the emulator with the game's code and data. These files are created by either specialized tools for game cartridges or regular optical drives reading the data. [1] Despite the increasing length of time between the discontinuation of the original system and product, most games retain their copyright, forcing regular consumers and emulation enthusiasts to resort to downloading games for free from various websites rather than purchasing and ripping the contents legally (although for optical media, this is not uncommon for legitimate owners). As an alternative, specialized adapters such as the Ret rode allow emulators to directly access the data on game cartridges without needing to copy it into a ROM image first.

Emulators can be designed in one of three ways: purely software, which is the most common, such as MAME using arcade ROM images; purely hardware, such as the ColecoVision's adapter to accept Atari VCS cartridges; or hybrid solutions, such as hardware bridgeboards for various Amiga computers that could run IBM PC-compatible software.

To avoid any potential conflicts with non-public intellectual property, emulators are typically created by reverse engineering hardware information. Some hardware specifications may be made public for developers to use in their emulation efforts, but there are often layers of information that remain as trade secrets, such as encryption details. Operating code stored in the BIOS of hardware can be disassembled and analyzed in a clean room setting, with one person disassembling and another documenting the code's function separately. An emulation on the target hardware can be built once enough information about how the hardware interprets the game software has been gathered. To avoid contaminating the clean room nature of their project, emulation developers typically avoid any information that may come from untraceable sources. In 2020, for example, a large trove of information about Nintendo's consoles was leaked,

and teams working on Nintendo console emulators like the Dolphin emulator for GameCube and Wii stated that they were staying away from the leaked information to avoid tarnishing their project.

Following the creation of an emulator, a copy of the game software must be obtained, a step that may have legal ramifications. This usually necessitates the user copying the contents of the ROM cartridge to computer files or images that the emulator can read, a process known as "dumping" the contents of the ROM. Other proprietary formats, such as PlayStation CD games, use a similar concept. While it is not required for emulation of the first arcade or home console, most emulators also require a BIOS dump, which varies depending on distribution region and hardware revisions. Emulators can sometimes be used to apply ROM patches, which update the ROM or BIOS dump to fix incompatibilities with newer platforms or change aspects of the game. The emulator then emulates the hardware using the BIOS dump, while the game software is replicated using the ROM dump (with any patches).

Although the primary purpose of emulation is to make older video-games execute on newer systems, there are several advantages inherent in the extra flexibility of software emulation that were not possible on the original systems.

- ROM hacking and modification
- Enhanced technical features
- Bypassing regional lockouts
- Cheating and widescreen functionality

Here is the list of some popular video game emulators

- FCEUX
- NESticle
- Nestopia
- AdriPSX

- bleem!
- bleemcast!
- Connectix Virtual Game Station
- ePSXe
- PCSX-Reloaded

4.3 ALGORITHM AND FLOWCHART

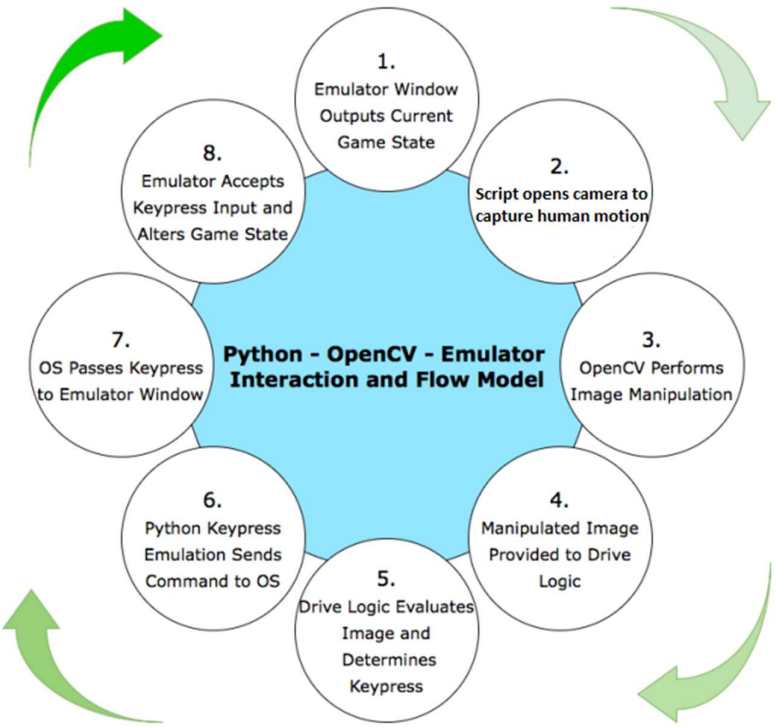

Fig. 4.4 Flowchart of the project

1. Initialize Emulator
   a. Initialize the emulator.
   b. Run the game from Game ROM.
2. Executing the program
   a. Opens camera to capture human motion
3. Image Manipulation
   a. Computer vision library OpenCV performs image Manipulation.
   b. It captures 20-30 frames per second.
4. Track Skeleton
   a. Skeleton frame ready method is used.
   b. As the user is continuously moving in front of camera, his movements are tracked.
5. Pose Detection
   a. Media pipe logic for processes the image for gesture detection.
6. Gesture Detection
   a. Once the gesture is detected the pyAutoGUI module comes in action.
7. Send Key press Event
   a. When a gesture is detected and threshold is matched, a particular gesture event is invoked.

4.3.1 MEDIAPIPE POSE

In applications like quantifying physical exercises, sign language recognition, and full-body gesture control, human pose estimation from video is critical. It can be used in yoga, dance, and fitness applications, for example. In augmented reality, it can also enable the overlay of digital content and information on top of the physical world.

MediaPipe Pose is a machine learning solution for high-fidelity body pose tracking that uses our Blaze Pose research to infer 33 3D landmarks and a background segmentation mask on the whole body from RGB video frames. For inference, most current state-of-the-art approaches rely on powerful desktop

environments, whereas our method achieves real-time performance on most modern mobile phones, desktops/laptops, in Python, and even on the web.

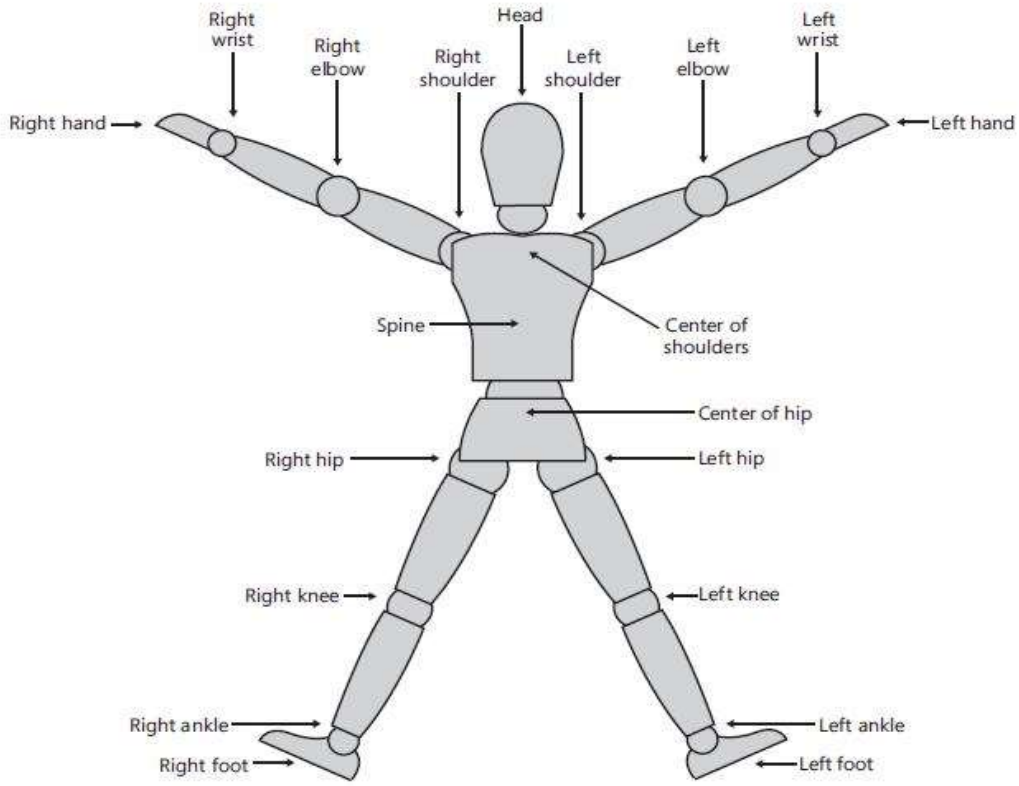

Fig. 4.5 Name of each point in the human body skeleton

Each skeleton joint is measured in a three-dimensional (X, Y, Z) plane. The X and Y coordinates specify the location of the joint in the plane, and the player facing the Kinect sensor is in the Z direction.

When a joint is represented with X, Y, and Z coordinates in a three-dimensional plane, the X and Y coordinates actually indicate the joint location in the plane, and Z indicates how far the joint is from the sensor. If the joints move from the right-hand side to the left-hand side or vice versa, the X axis of the joint will change. Similarly, for moving joints in the upwards or downwards direction, the value of the Y axis will change. Changes in the Z axis will reflect if the joints

move forward or backwards from the sensor. Calculations for the basic gestures can be done by either of the following:

- Calculating the distance between different joints
- Comparing the joints' positions and the deviation between the joints' positions

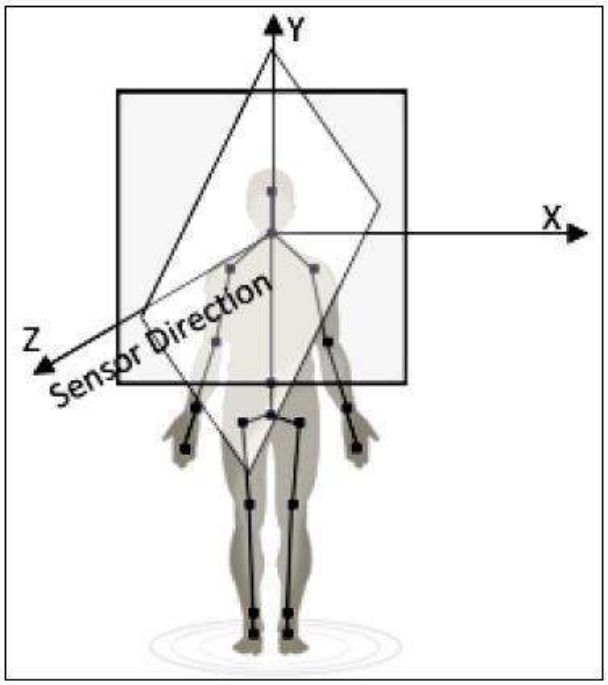

Fig. 4.6 Representation of X, Y, Z co-ordinates in 3- dimensional

4.3.2 CALCULATING THE DISTANCE BETWEEN TWO JOINTS

Skeleton data representation is three dimensional; however, before looking into the 3D coordinate plane, let's first consider the points in a 2D coordinate plane with only X and Y axis and see how to calculate the distance between two points. In general mathematics, to calculate the distance between two points, we need to make use of the Pythagorean Theorem. The theorem states that: For a right-angled triangle, the square of the hypotenuse is equal to the sum of the squares of the other two sides. Refer to the following diagram, which

shows how the Pythagorean theorem can be applied to calculate the distance between two joints:

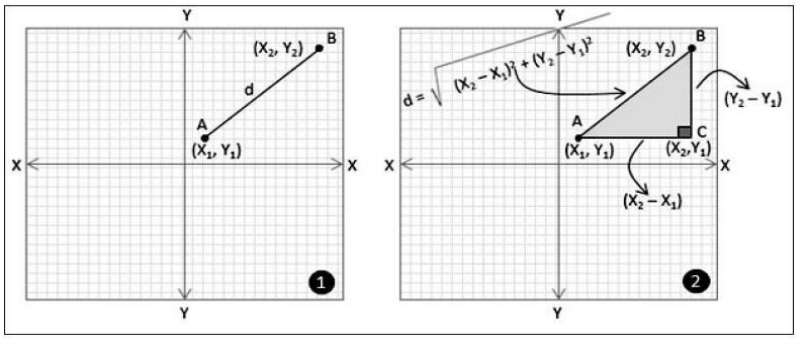

Fig. 4.7 Calculating the distance between two points to join

Consider that you have a point A (X1, Y1) and a point B (X2,Y2) in a two-dimensional coordinate plane. You want to calculate the distance (let's call it "d") between point A and point B (refer to the image marked as 1). To calculate the distance using the Pythagoras theorem we have to first draw a parallel line to the X axis from point A and another line from point B, which is parallel to the Y axis. Consider both the lines meeting at point C (X2, Y1). As we know, the X and Y axes are perpendicular to each other; the triangle formed by the points A, B, and C is a right-angled one. Now, the value of "d", the distance between points A and B, will be the hypotenuse of the right-angled triangle that was formed by the points A, B, and C. The distance between A and B, now can be calculated using following formula

$$d = \sqrt{(\text{distance of A,C})^2 + (\text{Distance of B,C})^2}$$

$$d = \sqrt{(X_2 - X_1)^2 + (Y_2 - Y_1)^2}$$

4.3.3 POSE LANDMARK MODEL

The landmark model in MediaPipe Pose predicts the location of 33 pose landmarks. MediaPipe Pose can predict a two-class segmentation mask for a full-body segmentation mask (human or background).

Let's break down the code by first understanding the function's functionality in a nutshell: This function will detect the pose of a person who is closely associated with the image and draw all of the landmark points in the pose that is detected in the image.

- image**:** This argument will demand the image in which the person is there to be detected.
- pose**:** The pose function which we created for images and video one of those will be required according to the requirements
- draw**:** This argument will have the Boolean value when the value is True which means the function should draw the landmarks points otherwise not.
- display**:** This argument will also expect the Boolean value and if it has the True value then it will show both the input image and the resultant image otherwise not.
- output image**:** It will return the input image with the landmarks points which were detected in the person’s pose.
- results: It shows the output on the input image i.e., it is responsible for the visibility of the output on the image

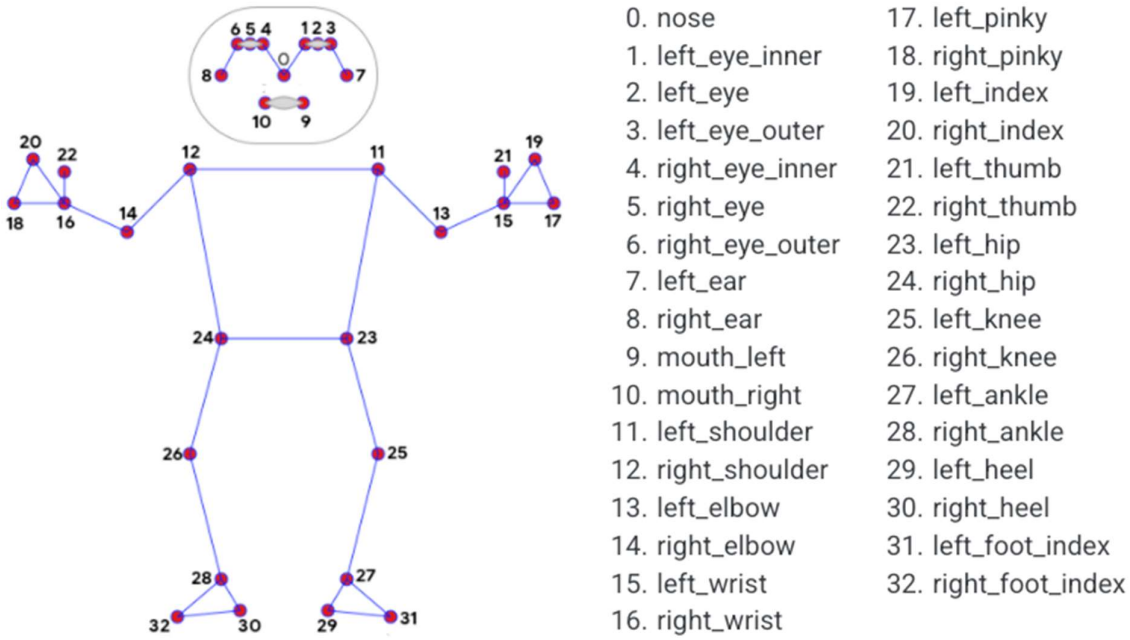

Fig. 4.8 Pose Landmark Model

CHAPTER – 5
TECHNICAL SPECIFICATION

CHAPTER – 5

TECHNICAL SPECIFICATION

5.1 ADVANTAGES:

Following are the advantages of our system:

1. Optimization of hardware:
2. Reduce hardware cost
3. User friendly
4. Portable
5. Elimination of high-priced consoles for playing motion sensed games.
6. Cheaper than current existing systems.
7. Can be used for system control also.
8. Affordable for normal users.

5.2 DISADVANTAGES

Following are the disadvantages of our system:

1. It can only track up to 1 player, so no 2 or more players can actively participate in a game.
2. Following are the disadvantages of standard camera:
3. Application can only detect pose only if there is good lighting.
4. Minimum length of 1m should be maintained between the player and the camera

5.3 APPLICATIONS

Following are the applications of our system:

1. To play motion sensed games on regular PC/Laptop by using only standard camera.
2. Physical handicapped person can control system through gestures.
3. Systems can be controlled remotely using our system.

CHAPTER – 6

CODING IMPLEMENTATION

CHAPTER – 6

CODING IMPLEMENTATION

The following is the coding implementation of the above proposed method. The code is implemented in python programming language using Anaconda's jupyter notebook.

6.1 IMPORTING THE REQUIRED LIBRARIES:

For this idea of creating the application, the need of some obvious libraries that are necessary.

1. OpenCV
2. MediaPipe
3. NumPy
4. PyAutoGui

As a result, make sure these are in your Python modules. If they aren't already installed, use pip install <Required Module> to quickly install them.

After you've installed all of the necessary libraries, import them as the first part of the script for creating this app.

```
import cv2
import numpy as np
import time

import pyautogui
import mediapipe as mp
```

6.2 SETTING UP CAMERA:

Creating a video capture class in the cap variable to read frames from the webcam. Then, using OpenCV's imshow () function, we read from that cap object, which returns the frame, and show it to the user. Also, if the user presses the escape key (the code for escape key is 27), all the windows created are

destroyed, which means the camera and application are closed. Basic code to reading frames from webcam and showing it to the user.

```
cap = cv2.VideoCapture(0)

while True:
      _, frm = cap.read()
      cv2.imshow("window", frm)

      if cv2.waitKey(1) == 27:
        cap.release()
        cv2.destroyAllWindows()
        break
```

6.3 MEDIAPIPE POSE DETECTION:

Pose object is being instantiated. Media pipe has a pose detection solution, and drawing is used to show the visuals to the user based on what the media pipe can detect. The pose object will now process our frame and return a result in frame. To process the frame, we use pose object. It takes a frame object as a parameter, but it reads it in RGB format, whereas OpenCV reads it in BGR format, so we convert it to RGB using CV2's cvtColor function.

```
cap = cv2.VideoCapture(0)
pose = mp.solutions.pose
pose_o = pose.Pose()
drawing = mp.solutions.drawing_utils

while True:
      _, frm = cap.read()
      cv2.imshow("window", frm)
      res = pose_o.process(cv2.cvtColor(frm, cv2.COLOR_BGR2RGB))

      if cv2.waitKey(1) == 27:
        cap.release()
        cv2.destroyAllWindows()
        break
```

To determine whether the user is in the frame or not, we will use the 0th, 16th, 15th, 24th, and 23rd coordinates from the fig 4.10 (i.e., landmark model). If these coordinates are in the frame, continue the process; otherwise, instruct the user to

ensure that his entire body is in the frame. And to draw pose landmark connections we use draw_landmarks function which takes frame as a parameter and pose landmarks from result and from pose object.

```
def inFrame(lst):
    if lst[24].visibility > 0.7 and lst[23].visibility > 0.7 and lst[15].visi-
bility>0.7:
        return True
    return False


while True:
      _, frm = cap.read()
      cv2.imshow("window", frm)
      res = pose_o.process(cv2.cvtColor(frm, cv2.COLOR_BGR2RGB))
      drawing.draw_landmarks(frm, res.pose_landmarks, pose.POSE_CONNECTIONS)
      if res.pose_landmarks and inFrame(finalres):
      //further process is done
      else: # here chek if any key down the make it up
      cv2.putText(frm, "Make Sure full body in frame", (50,100),
      cv2.FONT_HERSHEY_SIMPLEX, 1, (0,0,255), 2)
```

6.4 IS RUNNING LOGIC:

The straightforward to achieve this functionality is we will use 16th, 15th, 24th, and 23rd landmark position from fig 4.10(i.e., landmark model), we will add all four of these and we will have a previous sum and new sum then we compare these sums if they are greater than some threshold value, we can say that user has moved or is running otherwise the user is not running. This won't work for single frame so we have to analyze it from at least 3 or 4 frames. For that we are going to have an NumPy array of five length and we are going to initialize it with 0. Now this array works as a shifting and it stores the difference of previous sum and new sum and keeps shifting it by adding the new sum at first index. So, now to determine if the user is running, we can sum all of these elements and if the sum gets greater than some threshold value then the user is running otherwise the user is not running.

```
#logic
isInit = False
prevSum = 0
sum_list = np.array([0.0]*5)

def findSum(lst):
    sm = 0
    sm = lst[16].y*640 + lst[15].y*640 + lst[0].y*640 + lst[23].y*640 +
lst[24].y*640
    return sm

def push(sm):
    global sum_list
    for i in range(3, -1, -1): #reverse loop 3,2,1,0
        sum_list[i+1] = sum_list[i]

    sum_list[0] = abs(sm-prevSum)

def isRunning():
    sm = 0
    for i in sum_list:
        sm = sm + i

    if sm > 30:
        return True
    return False

while True:
      if res.pose_landmarks and inFrame(finalres):
        if not(isInit):
            prevSum = findSum(finalres)
            isInit = True
        else:
            newSum = findSum(finalres)
            push(newSum)

      if isRunning(): ##  running down the d key
                cv2.putText(frm, "Running", (50,100),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0,0,255), 2)
      else:
                cv2.putText(frm, "You are still", (50,100),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0,0,255), 2)
```

6.5 IS JUMPING LOGIC:

To achieve this functionality the logic the is quite simple and similar to previous one. The idea here is we check that the nose landmark i.e.,0^{th} landmark from landmark model if it goes above some threshold value then we are going to say that the user is jumping otherwise the user is not jumping.

```
def isJump(p):
    if p<80:
        return True
    return False


while True:
      if res.pose_landmarks and inFrame(finalres):

            ----

            if isJump(finalres[0].y*480):

            cv2.putText(frm, "JUMP DONE", (50,140), cv2.FONT_HERSHEY_SIMPLEX,
            1, (255,0,0), 2)
            else:
            cv2.putText(frm, "Not jump", (50,140), cv2.FONT_HERSHEY_SIMPLEX,
            1, (255,0,0), 2)

            ----
```

6.6 IS SHOOTING LOGIC:

To achieve this functionality the logic the is quite simple and similar to previous one. The idea here is we check that the 16th and 15th coordinate from landmark model are close to each other then we are going to say that the user is shooting otherwise the user is not shooting.

The logic used to achieve this functionality is quite simple and similar to the previous one. The idea is that if the 16th and 15th coordinates from the landmark model are close to each other, we will say that the user is shooting; otherwise, the user is not shooting.

```
def isShoot(finalres):
    if abs(finalres[15].x*640 - finalres[16].x*640) < 100:
        return True
    return False


while True:
      if res.pose_landmarks and inFrame(finalres):

            ____
            if isShoot(finalres):
            cv2.putText(frm, "Shooting", (50,180), cv2.FONT_HERSHEY_SIMPLEX,
            1, (85,0,85), 2)

            else:
            cv2.putText(frm, "Not Shooting", (50,180),
            cv2.FONT_HERSHEY_SIMPLEX, 1, (85,0,85), 2)
```

6.7 PYAUTOGUI KEYPRESS EVENTS:

The final step is to transfer all of the written logic into the game by converting the detected gestures. We use the pyAutoGUI library to implement this functionality, which handles keypress events. For instance, in order to move the character in the game, we must press the D key on the keyboard. As a result, we create a variable to store whether the current key is pressed or not. When the is Running logic is invoked, we must now invoke pyAutoGUI's key down events. Similarly, for is Jumping and is Shooting logic.

The final step is to convert the detected gestures and incorporate all of the written logic into the game. To implement this functionality, we use the pyAutoGUI library, which handles keypress events. To move the character in the game, for example, we must press the D key on the keyboard. As a result, we create a variable that stores whether or not the current key is pressed. When the is Running logic is called, we must now call the key down events of pyAutoGUI. Similarly, the logic for is Jumping and is Shooting. The code for the following is in the next page.

```
#d - move forward
#j - jump
#s - shoot
d_down=False
s_down=False
j_down=False

if res.pose_landmarks and inFrame(finalres):
        if not(isInit):
            prevSum = findSum(finalres)
            isInit = True
        else:
            newSum = findSum(finalres)
            push(newSum)

            if isJump(finalres[0].y*480): #press j down only if j is not already down
                cv2.putText(frm, "JUMP DONE", (50,140), cv2.FONT_HERSHEY_SIMPLEX, 1, (255,0,0), 2)

                if not(j_down):
                    pyautogui.keyDown("w")
                    j_down=True

            else: # relese j key
                cv2.putText(frm, "Not jump", (50,140), cv2.FONT_HERSHEY_SIMPLEX, 1, (255,0,0), 2)

                if j_down:
                    j_down=False
                    pyautogui.keyUp("w")

            if isRunning(): ##  running down the d key
                cv2.putText(frm, "Running", (50,100), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,0,255), 2)
                if not(d_down):
                    d_down = True
                    pyautogui.keyDown("d")

            else:
                cv2.putText(frm, "You are still", (50,100), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,0,255), 2)

                if d_down:
                    d_down = False
                    pyautogui.keyUp("d")

            if isShoot(finalres): # press s key for shooting
                cv2.putText(frm, "Shooting", (50,180), cv2.FONT_HERSHEY_SIMPLEX, 1, (85,0,85), 2)

                if not(s_down):
                    s_down = True
                    pyautogui.keyDown("z")
```

```
                    pyautogui.keyUp("z")

                prevSum = newSum


        else: # here chek if any key down the make it up
            cv2.putText(frm, "Make Sure full body in frame", (50,100), cv2.FONT_HERSHEY_SIMPLEX, 1,
(0,0,255), 2)

            if d_down:
                pyautogui.keyUp("d")
                d_down=False
            if s_down:
                pyautogui.keyUp("s")
                s_down=False
```

6.8 FINAL CODE:

```
import cv2
import numpy as np
import time
import pyautogui
import mediapipe as mp
pose = mp.solutions.pose
pose_o = pose.Pose()
drawing = mp.solutions.drawing_utils
cap = cv2.VideoCapture(0)
#logic
isInit = False
prevSum = 0
sum_list = np.array([0.0]*5)


#d - move forward
#j - jump
#s - shoot
d_down=False
s_down=False
```

```
j_down=False


def findSum(lst):
    sm = 0
    sm = lst[16].y*640 + lst[15].y*640 + lst[0].y*640 + lst[23].y*640 + lst[24].y*640
    return sm

def push(sm):
    global sum_list
    for i in range(3, -1, -1): #reverse loop 3,2,1,0
        sum_list[i+1] = sum_list[i]

    sum_list[0] = abs(sm-prevSum)

def isRunning():
    sm = 0
    for i in sum_list:
        sm = sm + i

    if sm > 30:
        return True
    return False

def inFrame(lst):
    if lst[24].visibility > 0.7 and lst[23].visibility > 0.7 and lst[15].visibility>0.7:
        return True
    return False

def isJump(p):
    if p<80:
        return True
    return False

def isShoot(finalres):
```

```
    if abs(finalres[15].x*640 - finalres[16].x*640) < 100:
        return True
    return False


while True:
    stime = time.time()

    _, frm = cap.read()

    res = pose_o.process(cv2.cvtColor(frm, cv2.COLOR_BGR2RGB))

    if res.pose_landmarks:
        finalres = res.pose_landmarks.landmark

    drawing.draw_landmarks(frm, res.pose_landmarks, pose.POSE_CONNECTIONS)

    cv2.putText(frm, f"{int(1/(time.time()-stime))}", (50,50),
cv2.FONT_HERSHEY_SIMPLEX, 1, (85,0,85), 2)

    #main logic
    if res.pose_landmarks and inFrame(finalres):
        if not(isInit):
            prevSum = findSum(finalres)
            isInit = True
        else:
            newSum = findSum(finalres)
            push(newSum)

            if isJump(finalres[0].y*480): #press j down only if j is not already down
cv2.putText(frm, "JUMP DONE", (50,140), cv2.FONT_HERSHEY_SIMPLEX, 1, (255,0,0),
2)

                if not(j_down):
                    pyautogui.keyDown("w")
```

```
                j_down=True

            else: # relese j key
                cv2.putText(frm, "Not jump", (50,140), cv2.FONT_HERSHEY_SIMPLEX, 1,
(255,0,0), 2)

                if j_down:
                    j_down=False
                    pyautogui.keyUp("w")

            if isRunning(): ##  running down the d key
                cv2.putText(frm, "Running", (50,100), cv2.FONT_HERSHEY_SIMPLEX, 1,
(0,0,255), 2)
                if not(d_down):
                    d_down = True
                    pyautogui.keyDown("d")
            else:
                cv2.putText(frm, "You are still", (50,100), cv2.FONT_HERSHEY_SIMPLEX, 1,
(0,0,255), 2)
                if d_down:
                    d_down = False
                    pyautogui.keyUp("d")
            if isShoot(finalres): # press s key for shooting
                cv2.putText(frm, "Shooting", (50,180), cv2.FONT_HERSHEY_SIMPLEX, 1,
(85,0,85), 2)

                if not(s_down):
                    s_down = True
                    pyautogui.keyDown("z")




            else: # release s key up
```

```
                cv2.putText(frm, "Not Shooting", (50,180), cv2.FONT_HERSHEY_SIMPLEX, 1,
(85,0,85), 2)

                if s_down:
                    s_down = False
                    pyautogui.keyUp("z")

            prevSum = newSum



    else: # here chek if any key down the make it up
        cv2.putText(frm, "Make Sure full body in frame", (50,100),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0,0,255), 2)

        if d_down:
            pyautogui.keyUp("d")
            d_down=False
        if s_down:
            pyautogui.keyUp("s")
            s_down=False


    cv2.line(frm, (0,80), (640,80), (255,0,0), 1)
    cv2.imshow("window", frm)

    if cv2.waitKey(1) == 27:
        cap.release()
        cv2.destroyAllWindows()
        break
```

CHAPTER – 7

SOFTWARE TESTING

CHAPTER – 7

SOFTWARE TESTING

7.1 INTRODUCTION TO TESTING

Testing is the process of assessing a system or its component(s) in order to determine whether it meets the specified requirements. The actual, expected, and difference between their results are the outcomes of this activity.

Simply put, testing is the process of running a system to find any gaps, errors, or missing requirements that are in conflict with the actual desire or requirements.

Software testing is the process of ensuring that a software application or programme works properly.

1. Meets the business and technical requirements that guided its design and development.
2. Works as expected.

Goals of Software Testing:

It is critical to determine the goals or objectives for which the application will be tested before proceeding.

- The foremost and basic intention behind Software testing is to make sure that the application under test is free of bugs and errors and in case it is not, it should be identified and sent for rectification.
- Software testing plays a vital role to improve the quality of the product/software by reporting bugs at different phases of development depending on the methodology followed for the project.
- Software testing brings with it a source of reliability that is important for any business to get hold of the market.
- Software testing brings with it a source of reliability that is important for any business to get hold of the market.
- System's Stability is one another checkpoint that Software Testing aims at since a stable application always gives rise to more customers and hence increases the revenue of the organization

- Software testing ensures whether the application is ready to deliver to the client or not, so it aims to find the critical bugs early in the testing phase.

Major types of testing:

White-Box testing**:** White-box testing (also known as clear box testing, glass box testing, transparent box testing, and structural testing) examines a program's internal structures and workings rather than the functionality that is visible to the end user. In white-box testing, test cases are created using an internal perspective of the system as well as programming skills. The tester selects inputs from which to exercise code paths and determine appropriate outputs.

Techniques used in our project's white-box testing include:

- Code Coverage– creating tests to satisfy some criteria of code coverage. All statements in the program to be executed at least once.

Black-box testing: Black-box testing treats the software as a "black box", examining functionality without any knowledge of internal implementation. The tester is only aware of what the software is supposed to do, not how it does it. Black Box Testing is a software testing method that involves testing the functionalities of software applications without knowing the internal code structure, implementation details, or internal paths. Black Box Testing is a type of software testing that focuses on the input and output of software applications and is completely based on software requirements and specifications. Behavioral testing is another name for it.

7.2 TEST CASES

In this section we are elaborating the testing methods implemented by us in this project.

- Unit Testing
- Integration Testing
- Functional Testing
- System Testing
- Stress Testing
- Performance Testing
- Usability Testing
- Acceptance Testing
- Regression Testing

1. Unit Testing

Unit testing is used to ensure that individual project components work as expected. The following items were completed in this section:

a. Testing the coded script.

b. Checking whether the camera is working properly or not.

c. Checking whether the human gesture detection is done properly or not.

2. Integration Testing

Integration testing is a type of software testing in which a software application's various units, modules, or components are tested as a single entity. However, different programmers may have created these modules.

Integration testing is used to test the interfaces between modules and expose any defects that may occur when these components are combined and must interact with one another. In this section we have combined the modules in the project and check whether we get desired output.

3. System Testing

With system testing we aim to test our system in various environments, since our system is meant to run on Windows platform, we ran the tests on Windows OS only.

According to our conclusions our system runs on following versions of Windows.

Operating System Version	Test Result	Requirements
Windows 8	System works as intended	. Python 3+
Windows 7	System works as intended	. Python 3+
Windows XP	System experiences time glitches as XP doesn't fully support Python 3.9	-

Table 7.1 System Testing for various operating systems

4. Stress testing

In this section we evaluate how system behaves under unfavorable conditions. Here we try outing different combinations of actions and interpreting our results.

Only 1 Human can be detected by the system at a time, if some extra human comes in range of the camera, the camera still detects one who is more clearly visible. We have planned to eliminate this in the next release.

5. Usability Testing

Usability testing is performed to the perspective of the client, to evaluate how the GUI is user-friendly; this is part of black-box testing. Here in this project, as a part of usability testing, we have found the GUI to be user friendly. We have asked for input from neutral persons related to the GUI and found that the GUI is self-

explanatory and the user finds it easy to understand functionality of the project from GUI.

6. Functional Testing

Functional testing is the testing to ensure that the specified functionality required in the system requirements works. It falls under the class of black box testing.

- Check for different functions involved in the code as parsing, encrypting data, secret sharing, decrypting, and lastly reuniting the fragments in order to achieve the retrieval of original data back.

7. Performance Testing

Performance testing is the testing to assess the speed and effectiveness of the system and to make sure it is generating results within a specified time as in performance requirements. It falls under the class of black box testing.

- Providing several no. of inputs in order to check for its correctness, efficiency, and what much time it takes to produce the result i.e., retrieval of original data.

6. Acceptance Testing

Acceptance testing is often done by the customer to ensure that the delivered product meets the requirements and works as the customer expected. It falls under the class of black box testing.

9. Regression Testing

Regression testing is the testing after modification of a system, component, or a group of related units to ensure that the modification is working correctly and is not damaging or imposing other modules to produce unexpected results. It falls under the class of black box testing.

After once the system is used, its effectiveness will be studied…and if there exist any modification to be done, to increase the system efficiency, in that case algorithms will be again studied.

CHAPTER – 8

RESULTS

CHAPTER – 8

RESULTS

Let's see what we have achieved in this project till now:

1) At the moment there is no system developed for playing all existing PC games Camera on Windows platform, with our software we have achieved this. Hence, we are bringing motion sensed gaming on Windows platform via deep learning.
2) In our system we are able to play games which rely on following Keys: D to move forward, S to shoot, J to jump.

- When nobody is in the frame

Fig. 8.1 When nobody is in the frame

- Is Still

Fig. 8.2 is Still

- Is Running

Fig. 8.3 is Running

- Is Jumping

Fig. 8.4 is Jumping

- Is Shooting

Fig. 8.5 is Shooting

3) We have successfully detected human gestures/actions and when these gestures are matched key press event is invoked.

4) On Commercial point of view, we can release gesture sets for games who wish to enhance their current gaming experience on computers.

CHAPTER – 9

DEPLOYMENT

CHAPTER – 9

DEPLOYMENT

9.1 FCEUX EMULATOR

There is no need to download any installation files; simply extract and run. Visit https://fceux.com/web/download.html to get the FCEUX emulator. then, depending on your system architecture, download the most recent release.

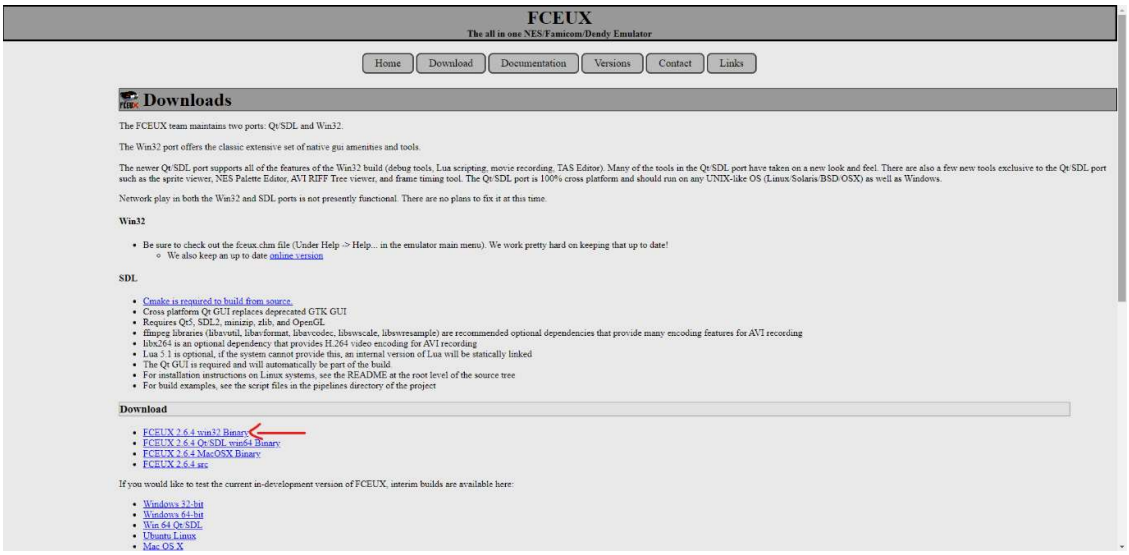

Fig. 9.1 Downloading FCEUX Emulator

Once the file has been downloaded, go to the downloaded folder and extract the file. then navigate to the fceux-2.6.4-win32/bin directory. To use the emulator, open the qfceux application.

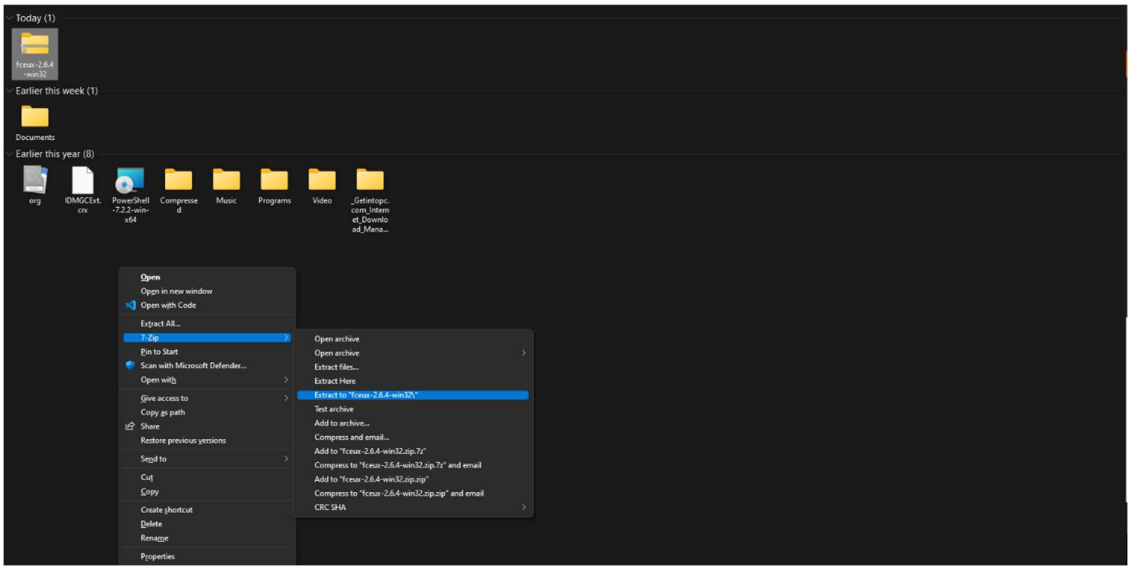

Fig. 9.2 Extracting the emulator

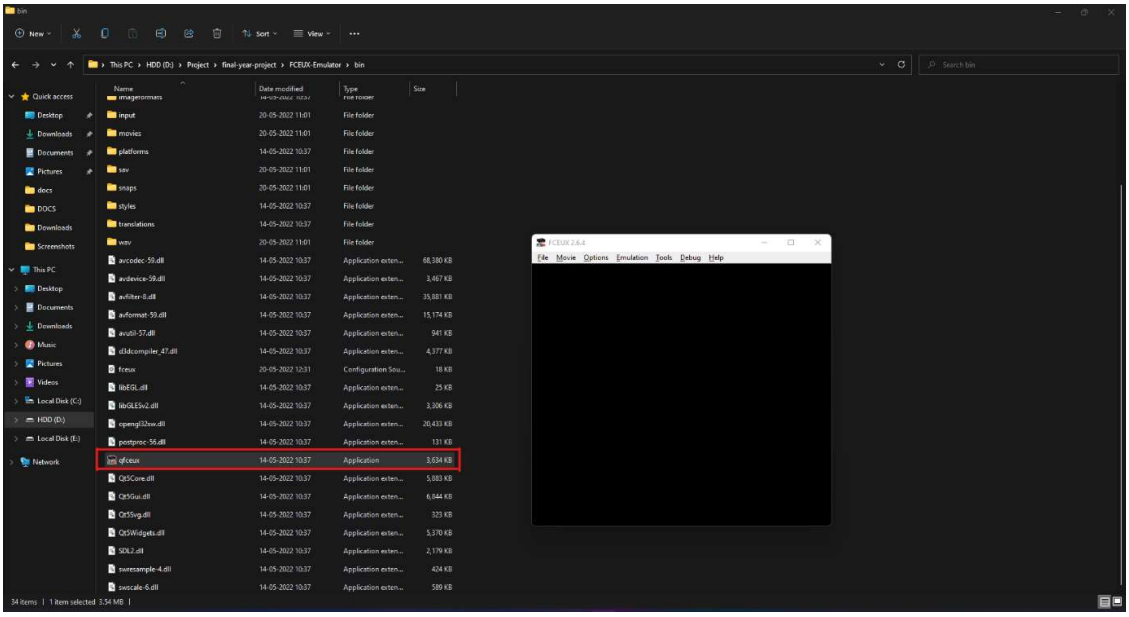

Fig. 9.3 Running the emulator

9.2 OPENING THE ROM FILE AND EXECUTING THE SCRIPT

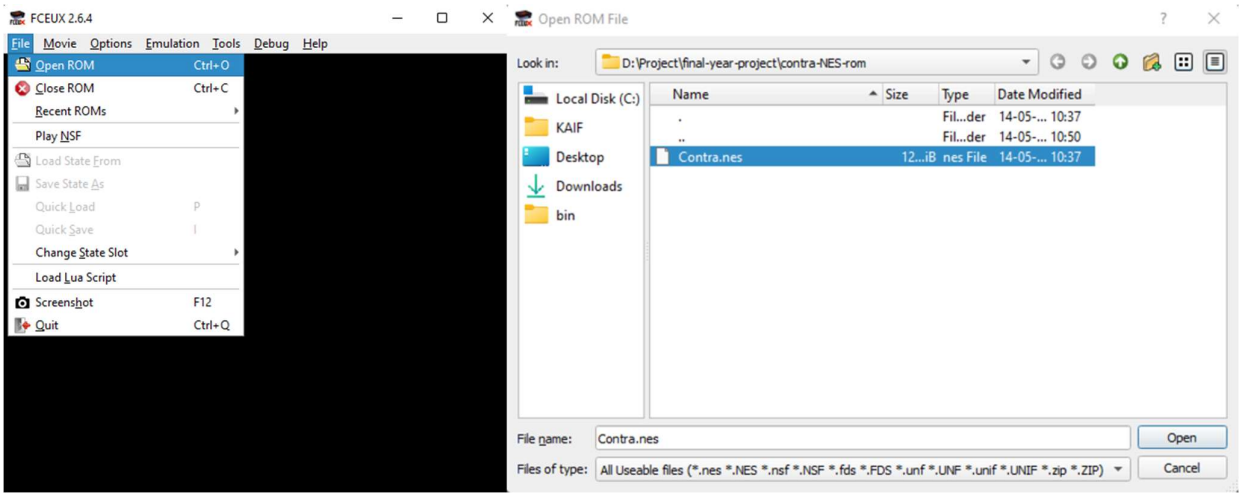

Fig. 9.4 Opening the ROM File

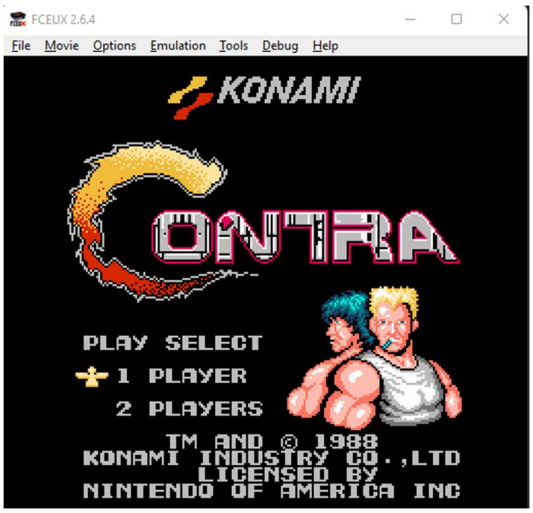

Fig. 9.5 Running the ROM file

To Open the Rom file:

- Select file.
- Click on Open ROM.
- Navigate to the rom file saved in the computer.
- To download the rom files, you can visit any of the websites like EMU paradise to get the rom files
- Finally run the python script.

CHAPTER – 10

CONCULSION AND FUTURE SCOPE

CHAPTER – 10

CONCLUSION AND FUTURE SCOPE

10.1 CONCLUSION

Hence in this project we are eliminating high end console systems and replacing it with our regular PC/Laptop's integrated camera also we are controlling games without actually touching keyboard or mouse.

10.1 FUTURE SCOPE

The frame-processing rate needs to be increased as a major improvement in the project. Even as the number of pyramid levels grows, processing should improve, and detection should occur at a rate of as little as 0.4 frames per second. This will be accomplished by using GPU to perform some calculations. Furthermore, focus is required. The retained mode will be implemented with great effort. Human detection consistency must also be improved, particularly in far-plane environments. The consistency can be improved by performing as many tests as possible in different locations and thus achieving the relationship between detection and threshold values. Because the frames are drawn in real time, the focus is on the OpenGL drawing routine. Finally, to add interest to the game, some sound effects or even background music could be added.

CHAPTER – 11

REFERENCES

BIBLIOGRAPHY

[1] Chia-Chih Chen; Aggarwal, J.K..: Human Detection Using Depth Information by Kinect. CVPR, 1 (2011) 886-893.

[2] T. Darrell, G. Gordon, J. Woodfill, and M. Harville, "Integrated Person Tracking using Stereo, Color, and Pattern Detection," Proceedings IEEE Conference on Computer Vision and Pattern Recognition, Santa Barbara

[3] http://www.KinectProgrammingGuide.com/human_trackingusing_hardware/

[4] http://www.codeplex.com/kinect/gesture_detection/

[5] https://google.github.io/mediapipe/

[6] https://pyautogui.readthedocs.io/en/latest/

[7] N. Dalal and B. Trigges.: Histograms of oriented gradients for human detection. CVPR, 1 (2005) 886-893.

[8] N. Dalal, B. Triggs, C. Schmid.: Human detection using oriented histograms of flow and appearance, in: European Conference on Computer Vision, Graz, Austria, May 7–13, 2006

[9] T. Darrell, G. Gordon, J. Woodfill, and M. Harville, "Integrated Person Tracking using Stereo, Color, and Pattern Detection," Proceedings IEEE Conference on Computer Vision and Pattern Recognition, Santa Barbara, June 1998

[10] D. Demirdjian, T. Darrell: 3-D Articulated Pose Tracking for Untethered Diectic Reference. ICMI 2002: 267-272

[11] V. Ganapathi, C. Plagemann, D. Koller, S. Thrun. Real time motion capture using a single time-of-flight camera. Proceedings of CVPR 2010. pp.755~762

[12] S. Ikemura, H. Fujiyoshi.: Real-Time Human Detection using Relational Depth Similarity Features. ACCV 2010, Lecture Notes in Computer Science, 2011, Volume
6495/2011, 25-38

[13] HP. Jain and A. Subramanian. Real-time upper-body human pose estimation using a depth camera. In HP Technical Reports, HPL-2010-190, 2010

[14] K. Levi and Y. Weiss. Learning object detection from a small number of examples: the

importance of good features. CVPR 2(2004) 53-60

[15] DG. Lowe.: Object Recognition from Local Scale-Invariant Features. Proceedings of

the International Conference on Computer Vision. 2 (1999). pp.?1150–115