



Faculty of Electrical Engineering

Department of Electronics and Information Technologies

# MASTER THESIS

Realization of virtual home assistant based on computer vision technology

Author: Bc. Andrei Gudovich

Supervisor: Ing. Ivo Veřtát, Ph.D.

Pilsen 2023

ZÁPADOČESKÁ UNIVERZITA V PLZNI

Fakulta elektrotechnická  
Akademický rok: 2022/2023

# ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení: **Bc. Andrei GUDOVICH**  
Osobní číslo: **E20N0060P**  
Studijní program: **N0714A060013 Elektronika a informační technologie**  
Specializace: **Informační a komunikační technologie**  
Téma práce: **Realizace virtuálního domácího asistenta na základě technologie počítačového vidění**  
Zadávací katedra: **Katedra elektroniky a informačních technologií**

## Zásady pro vypracování

1. Design an implementation of a virtual assistant based on computer vision for controlling a smart home.
2. Implement selected functions to demonstrate the capabilities of the virtual assistant for controlling a smart home.
3. Discuss the possibilities of using a virtual assistant for aging people living alone in the home and monitoring their life activities.

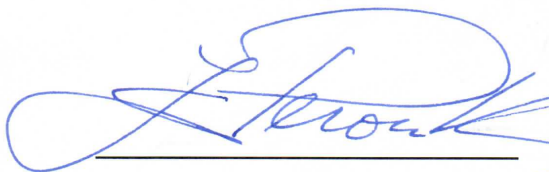
Rozsah diplomové práce: **40 – 60**  
Rozsah grafických prací:  
Forma zpracování diplomové práce: **elektronická**  
Jazyk zpracování: **Angličtina**

Seznam doporučené literatury:

1. VOBORNÍK, A., VEŘTÁT, I., LINHART, R. Experimental electric power system for small satellites with independent supply channels. In International Conference on Applied Electronics (AE 2018) : /proceedings/. Pilsen: University of West Bohemia, 2018. s. 155-161. ISBN: 978-80-261-0721-7 , ISSN: 1803-7232.
2. GANSSELE, J. A Designer's Guide to Watchdog Timers. Accessible online - <https://www.digikey.com/en/articles/a-designers-guide-to-watchdog-timers>

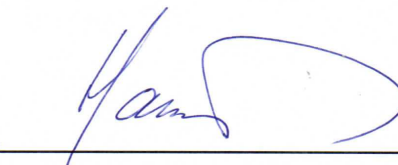
Vedoucí diplomové práce: **Ing. Ivo Veřtát, Ph.D.**  
Katedra elektroniky a informačních technologií

Datum zadání diplomové práce: **7. října 2022**  
Termín odevzdání diplomové práce: **26. května 2023**



L.S.

**Prof. Ing. Zdeněk Peroutka, Ph.D.**  
děkan



**Doc. Ing. Jiří Hammerbauer, Ph.D.**  
vedoucí katedry

V Plzni dne 7. října 2022

# Abstrakt

Tématem této práce je prostudovat možnost využití technologie počítačového vidění pro rozšíření funkčnosti virtuálního domácího asistenta pro chytrou domácnost.

Práce popisuje proces tvorby virtuálního asistenta pro chytré domácnosti v kontextu konceptu Ambient Assisted Living (AAL) pro seniory. AAL zahrnuje používání zařízení a způsobů, jak zajistit, aby starší lidé byli schopni dožít v domově v bezpečí s podporou monitorovacích systémů. Tyto technologie usnadňují život seniorů a do určité míry zvyšují jejich samostatnost.

Hlavními funkcemi implementovaného virtuálního asistenta jsou sběr statistických informací o činnosti uživatele, zjišťování jeho stavu a reakce na mimořádné události. Mezi další funkce patří možnost ovládat zařízení chytré domácnosti pomocí gest.

Pro polohu těla, aktivitu a rozpoznávání gest se používá technologie hlubokého strojového učení. Některé úlohy, jako je určování polohy těla a rukou, jsou řešeny pomocí předem natrénovaných neuronových sítí z frameworku Google Mediapipe. Specifičtější úkoly, jako je určování gest rukou a pozice uživatele, jsou řešeny pomocí neuronových sítí trénovaných na datech shromážděných během práce na projektu. Speciální pozornost je věnována procesu výběru architektury neuronových sítí s přihlédnutím k řešeným úlohám a porovnávání jejich metrik.

Výsledkem práce, který je uveden v závěru, je program virtuálního asistentu založený na technologii počítačového vidění ve fázi "proof of concept", který splňuje uvedené funkční požadavky.

V závěru jsou diskutovány potenciální aplikace pro vyvinutý systém, prozkoumána možnost jeho zlepšení a modernizace a představené možnosti použití projektu v jiných oblastech.

## Klíčová slova

Počítačové vidění, Domácí asistent, Asistované bydlení, Google Mediapipe, Odhad polohy těla, odhad gest, neuronová síť.

# Abstract

The topic of this work is to study the possibility of using computer vision technology to expand the functionality of a virtual home assistant for a smart home.

The thesis describes the process of creating a virtual assistant for smart homes in the context of the concept of Ambient Assisted Living (AAL) for the elderly. AAL involves the use of technologies and ways to ensure that the older persons stay safe and are able to age in place. These technologies makes lives of the older persons easier and to some extent, self-dependent.

The main functions of the implemented virtual assistant are the collection of statistical information about the user's activity, determining his state and responding to emergencies. Additional functionality includes the ability to control smart home devices using gestures.

For body position, activity and gesture recognition deep machine learning technology is used. Some tasks, such as determining the position of the body and hands, are solved using pre-trained neural networks from the Google Mediapipe framework. More specific tasks, such as determining hand gestures and user pose are solved by using neural networks trained on data collected during the project. Special attention is paid to the process of choosing the architecture of neural networks, taking into account the tasks being solved and to comparing their metrics.

The result of the work, which is presented in the conclusion, is a virtual assistant program based on computer vision technology at the "proof of concept" stage that meets the stated functional requirements.

The conclusion also delves into potential applications for the developed system, explores avenues for improvement and modernization, and examines the feasibility of applying the project's advancements in other domains.

## Keywords

Home assistant, Ambient Assisted Living, Computer vision, Google Mediapipe, Neural network, Pose estimation, Gesture estimation.

## Prohlášení

Předkládám tímto k posouzení a obhajobě diplomovou práci, zpracovanou na závěr studia na Fakultě elektrotechnické Západočeské univerzity v Plzni.

Prohlašuji, že jsem svou závěrečnou práci vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 270 trestního zákona č. 40/2009 Sb.

Také prohlašuji, že veškerý software, použitý při řešení této diplomové práce, je legální.

V Plzni dne 26. května 2023

Bc. Andrei Gudovich

.....

Podpis

# List of contents

<b>List of figures</b>	<b>vii</b>
<b>List of symbols and abbreviations</b>	<b>viii</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Definition of functional requirements</b>	<b>4</b>
2.1 Activity tracking . . . . .	4
2.2 Fall detection . . . . .	5
2.3 Smart home device control . . . . .	5
<b>3 Technical solution preparation</b>	<b>7</b>
3.1 General logic of the system . . . . .	7
3.2 Chose of the technologies . . . . .	8
3.2.1 Activity tracking . . . . .	8
3.2.1.1 Determination of the position of the human body . . . . .	8
3.2.1.2 Determination of user location . . . . .	13
3.2.2 Fall detection . . . . .	14
3.2.2.1 Determination of the position of the human body . . . . .	14
3.2.2.2 Fall detection . . . . .	14
3.2.2.3 Summary . . . . .	16
3.2.3 Smart home device control . . . . .	17
3.2.3.1 Selecting devices for interaction . . . . .	17
3.2.3.2 Determining the position of the hands . . . . .	18
3.2.3.3 Gesture command classification . . . . .	19
3.2.3.4 Sending a command to a device . . . . .	20
3.2.3.5 Summary . . . . .	20
3.2.4 Camera requirements . . . . .	20
<b>4 Technical implementation</b>	<b>22</b>
4.1 Receiving video input . . . . .	23
4.2 Activity tracking . . . . .	23
4.2.1 Divide camera frame into zones . . . . .	24

4.2.2	Activity tracking . . . . .	24
4.2.2.1	Determination of user location . . . . .	26
4.3	Fall detection . . . . .	27
4.3.1	Determining the position of the user's body . . . . .	27
4.3.2	Fall detection . . . . .	27
4.3.2.1	Pose classification . . . . .	27
4.3.2.2	Excluded zones . . . . .	41
4.4	Smart home device control . . . . .	42
4.4.1	Device selection . . . . .	42
4.4.2	Determining the position of the user's hands . . . . .	46
4.4.3	Hand gestures classification . . . . .	47
4.4.4	Communication with devices . . . . .	49
<b>5</b>	<b>Conclusion</b>	<b>50</b>
	<b>Reference</b>	<b>53</b>



# List of figures

3.1	Block scheme of the general program logic. . . . .	8
3.2	Human pose models. . . . .	9
3.3	Human pose estimation pipeline overview. . . . .	12
3.4	Vitruvian man aligned via two virtual keypoints predicted by BlazePose detector in addition to the face bounding box. . . . .	13
3.5	Tracking network architecture: regression with heatmap supervision. . . . .	13
3.6	MsediaPipe Pose landmarks. . . . .	13
3.7	21 hand landmarks. . . . .	19
3.8	Aligned hand crops passed to the tracking network with ground truth annotation. Bottom: Rendered synthetic hand images with ground truth annotation. . . . .	19
4.1	Block scheme of the virtual smart home assistant. . . . .	22
4.2	Body key points superimposed on video stream. . . . .	26
4.3	Examples of normal and unnatural pose. . . . .	29
4.4	Neural network comparison with varying numbers of neurons in the hidden layer. . . . .	32
4.5	Comparison of single-layer and two-layer neural networks. . . . .	33
4.6	Comparison of different activation functions of inner layers. . . . .	34
4.7	Comparison of different activation functions of output layer. . . . .	35
4.8	Comparing neural networks with varying regularization types and coefficients. . . . .	36
4.9	Comparing different optimizers with Learning Rate 0.001. . . . .	36
4.10	Comparing different optimizers with optimized Learning Rates. . . . .	37
4.11	Comparing different Learning Rates. . . . .	38
4.12	Comparing different batch sizes. . . . .	39
4.13	Confusion matrix. . . . .	40
4.14	Examples of normal and unnatural pose classification. . . . .	41
4.15	Pointing beam superimposed on the output video. . . . .	43
4.16	Device selection by the pointing beam. . . . .	43
4.17	Example of the functioning of the pointing movement filter. . . . .	45
4.18	Pointing beam snapped to the device bounding box. . . . .	46
4.19	Hands key points. . . . .	47

4.20	Examples of gestures for smart home devices control. . . . .	48
4.21	Examples of gesture classification. . . . .	49

# List of symbols and abbreviations

NN .....	Neural Network.
HPE .....	Human Pose Estimation.
CV .....	Computer Vision.
NLP .....	Natural Language Processing.

# 1

## Introduction

The theme of this project is to study the possibility of using computer vision technology in everyday life, namely the possibility of its integration with a virtual home assistant for the elderly.

In recent years, due to the growth in the quality of medicine and the constantly improving the standard of living, there has been a clear trend towards an increase in the average age of the world's population. This trend is especially pronounced in developed countries. One of the consequences of this demographic process is an increase in the stratum of the population of retirement age [3] [4].

One of the most significant problems that the increase in the average age of the population entails is the problem of caring for the elderly living separately. Due to the cognitive, physiological and physical changes in the body associated with age, a significant number of older people are faced with the need for regular monitoring of their condition and the state of their living conditions. Some of these tasks are covered by the relatives of the elderly person, but often these responsibilities fall on social services. However, family supervision and supervision by social services may not be enough. Even in the case of daily visits to an elderly person by relatives or social workers, these visits are usually limited from few minutes to several hours. At the same time, the older the person, the higher the likelihood of incidents requiring immediate response, therefore, the need for constant monitoring increases.

In the last decade, in an attempt to solve this problem, the use of modern technologies has increased significantly. One of the technological concepts aimed at improving the quality of life and safety of the elderly is Ambient Assisted Living (AAL). AAL environment can be defined as "the use of information and communication technologies in a person's daily life to enable them to stay active longer, remain socially connected, and live independently into old age" [1]. AAL can provide an array of solutions for improving the quality of life of individuals, for allowing people to live healthier and independently for longer, for helping people with disabilities, and for supporting caregivers and medical staff [2].

As mentioned above, aging leads to a gradual physical and physical decline, which in-

creases dependence on other people. Starting from this consideration, two different aspects can be examined. On the one hand, new technologies can be used to monitor older people in order to reduce daily physical assistance and prolong their autonomous life. In this context, medicine and diet adherence, sleep monitoring, or fall detection are some of the main issues that are to be considered and tackled. On the other hand, monitoring people who are not yet very old and that do not suffer from particular pathologies is strategic for the purpose of detecting eventual lifestyle changes and suspicious behaviors that could warn of the onset of neurodegenerative diseases at a very early stage [2].

Based on the foregoing, it was decided to explore the possibility of using computer vision technology to create such a virtual home assistant that can solve some of the problems associated with caring for the elderly, improving their quality of life and safety level. To date, there are many assistant programs based on natural language processing (NLP) technology, such as Google Assistant, Alexa from Amazon and Alisa from Yandex, which use smart speakers as a hardware platform, such as Google Nest, Amazon Dot, Yandex Station and others. These products allow you to combine different smart home devices into one system and control them using voice commands.

The main advantage of voice assistants is the great variability and rather high complexity of the commands they perceive. Modern voice assistants are able to quite accurately understand logically complex expressions and conditions. Also, a serious advantage of voice assistants is user feedback. This allows to exchange information in a two-way mode, answer user questions, provide him with the requested information, inform about problems or the inability to execute a command.

So, what benefits can a computer vision-based home assistant have over NLP-based solutions? The most obvious advantage of CV solution is the ability to collect information not related to sound. This allows the assistant to perform various actions without commands at all. For example, such a system can automatically turn on the light in a room when a person enters and turn it off when no one is in the room. Similarly, such a home assistant can turn off the power of some electrical appliances if it decides that they are left unattended.

Another opportunity that CV technology opens up is the collection of additional information about the behavior and state of users. Such a system can track the everyday activity and movement of a person around the home and store this information for later processing. Based on this data, certain patterns in user behavior can be identified. These information can then be used to diagnose neurodegenerative diseases at an early stage by abrupt changes in user's behavior. For example, a person may reduce the number of meals or regularly stay in bed longer than usual after waking up. Such changes in behavior may indirectly indicate a decline in human health.

Also, thanks to computer vision technology, it becomes possible to identify unusual or unnatural postures and positions of the user in space. For example, if a person lies motionless on the floor for a while, this is a clear signal of an emergency or critical

condition of the user. The system can immediately respond to this event and call for help.

Also, such a system can take on some of the simple actions implemented in NLP-based systems, namely the control of smart home devices at the request of the user. It should be mentioned that the ability of a computer vision assistant to understand commands will be limited compared to the capabilities of a voice assistant. The bottleneck of a computer vision assistant is the relatively small variability in well-defined gestures that can be associated with commands.

Another area of use of computer vision technology in the implementation of a home assistant is to improve security. Firstly, such a home assistant can register emergencies such as smoke, fire or flooding and inform the user and special services about this. Secondly, in such a system it is possible to implement a security function that will be able to determine the penetration of unauthorized persons into the premises and immediately notify the user.

Although computer vision technology is inferior to the voice assistant in terms of two-way interaction and the number of perceived commands, it can be used in special cases. Such a system can solve the problem of interacting with a smart home for hearing-impaired, deaf, deaf-mute people or people with speech impairments.

Assuming that this virtual assistant will be used mainly by the elderly it can be supposed that the problem of the low complexity of commands given using gestures will be secondary. According to statistics [5], [6], older people are less active in using technology and the Internet, so a limited set of commands for managing smart home devices will cover most of their needs. In addition, a computer vision-based system can be integrated with a natural language processing-based home assistant and combine their functionality. At the same time, the strengths of computer vision technology fit well with the needs of this target group. For example, the ability to monitor the status of electrical appliances and turn them off if necessary can be quite useful function, taking into account the correlation of deterioration in memory and attention of a person with age [7], [8], [9]. The ability to analyze the behavior and state of the user will allow to remove part of the burden from social workers caring for the elderly and reduce the expenses of the social services.

## 2

# Definition of functional requirements

Based on the concept described in the introduction, the functional requirements for the implementation of a virtual home assistant for the elderly will be drawn up.

This chapter will list the general requirements for virtual assistant functionality and explain why these features were considered to be important. The development of the logic of the program, the decomposition of the tasks and the development of technical solutions to problems will be described in the following chapter. Virtual assistant functionality requirements will be identified and described in order of their importance to project implementation.

## 2.1 Activity tracking

The first task is to provide the ability to track the daily activity of the user in order to collect data for statistical processing. This functionality considered to be important because it allows to determine the change in the state of the user or identify an emergency by changes in habitual patterns of behavior. For example, the user may stay in bed for a long time or stop cooking. Such changes in behavior may indirectly indicate a deterioration in human health.

Tracking daily activity within the framework of this project will mean tracking the user's movements around the home and recording the time spent in different functional areas of the premises. To implement this functionality, the field of view of the cameras in the premises will be divided into areas corresponding to the functional areas of the premises, for example, "kitchen", "dinner table", "workplace", "bed".

The presence of the user in a certain area will be determined based on information about the position of his body in the field of view of the camera relative to the outlined areas. To do this, the coordinates of the user's body will be compared with the coordinates of these zones.

In order to meet these requirements, the the program should:

- track the position of the user's body in space;

- determine the presence of the user in certain areas outlined in the field of view of the camera;
- keep track of the time spent by the user in each zone;
- allow to manually divide the camera view area into zones;
- save collected data for subsequent statistical processing;

## 2.2 Fall detection

The second task is to be able to track the position of the user's body and classify their unnatural states. This function is one of the key features to improve the safety of the elderly, living separately, as it allows them to identify the critical state of the user and immediately report it.

To indirectly determine the human condition using computer vision technology, many different methods and indicators can be used, however, within the framework of this project, the author will limit himself to simplified functionality. An unusual state will be taken as the case when the user lies on the floor or in another place not intended for this for some time. In this case, the virtual assistant must register this event and report it.

The horizontal position of the user's body, which would be considered unnatural, is not always an indicator of a critical situation. For example, the user may lie on a bed. To prevent false alarm, excluded zones will be highlighted in the camera field of view, in which the horizontal position of the body will not be classified as unnatural.

So, based on foregoing, the final version of the virtual assistant must:

- track the position of the user's body in space;
- classify the user's unnatural pose;
- provide a tool to manually define places where lying would not be classified as an unusual condition;

## 2.3 Smart home device control

The third task is to implement the ability to control smart home devices using gestures. The purpose of this function is to simplify the performance of daily activities for an elderly person. For example, the user will be able to remotely turn on or switch off light, adjust its brightness, adjust the temperature of the air conditioner, etc.

For comfortable control of smart home devices, it is necessary to provide for such nuances as a simple and intuitive way to select a device for interaction and a way to choose a command for a device.



The easiest and most natural way to select a smart home device to interact with is to point to the device with a hand. To implement this function, it is necessary to have data on the position of the user's arms.

In order for a smart home device to be selected by pointing to it, it is necessary to provide the program with information about the location of these devices in the frame. One of the simplest and most reliable ways to do this is to mark these devices in the frame manually with bounding boxes.

To select a command for a device gestures can be used. At the same time, a sufficient number of clearly distinguishable simple gestures should be provided to cover the basic functions of most devices, such as "Turn on", "Turn off", "Adjust". In order to implement this functionality, the program needs to receive data on the position of the user's hands. Based on this data, the gestures responsible for various commands can be classified. After the gesture has been classified, a command can be generated based on it, and transmitted to the device.

To implement this functionality, the virtual assistant must:

- track the position of the user's arms;
- track user's hands position;
- based on information about the position of the hands of the user, classify the gesture commands;
- provide the minimum required set of gesture commands: turn on, turn off, adjust;
- allow to manually mark smart home devices in the field of view of the camera;
- provide the interaction of the virtual assistant with device via API from the devices manufacturer;

# 3

## Technical solution preparation

### 3.1 General logic of the system

Based on the functional requirements formulated in the previous chapter, let's define the general logic of the implemented virtual assistant.

The operation of the system begins with receiving input data in the form of a video stream from the camera.

Having received the video, several tasks can be solved in parallel. The video stream can be used simultaneously to determine the position of the user's body in space and determine the position of the user's hands.

The primary task on the basis of which the main functionality will be implemented is to determine the position of the human body. To do this, the input video stream must be processed by an algorithm that will determine the presence of a person in the frame and determine its position.

Having solved this problem and received data about the position of the user's body, we can simultaneously solve tasks that need this data.

The first of them is to track user activity. To solve this task, the camera's field of view will be divided into areas, for example: "workplace", "bed", "kitchen". Based on the information about the position of the key points of the user's body in the frame, the program will determine in which area the person is located and take into account the time spent in each of the areas. This data will be regularly saved to a separate file for later use and statistical processing.

At the same time data about the position of the user's body will be passed to the input of the block, which will classify whether the position of the body is unnatural. If an unnatural pose is detected outside the safe area, a countdown will be started after which will be called a command to take appropriate action. When the user returns to the normal position, the time will be reset to zero.

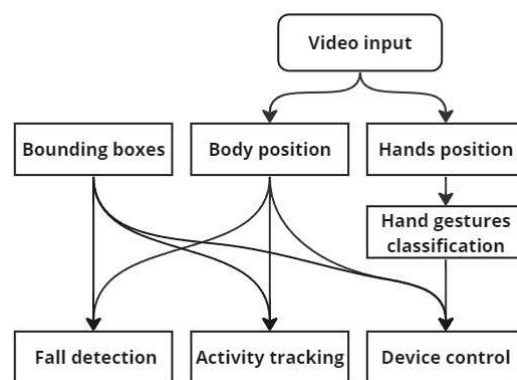
Also, based on the body position data, the user's intention to interact with smart home devices will be determined by pointing to a device with a hand. To do this, smart home devices that are in the field of view of the camera will be highlighted with rectangles. The

choice of a device for interaction will occur when the beam passing through the user's elbow and wrist will intersect the rectangle related to the device. To calculate the beam and its intersection with the rectangles in the frame, a module responsible for geometric calculations will be created.

When registering a request to interact with the device, tracking of the position of the user's hands will be started. Information about the position of the hands will be used to determine the gesture command for the selected smart home device.

After receiving the command for a selected device, the system will contact the device via the manufacturer's API and send the command to it.

Simplified block scheme of the general program logic is represented on the fig. 3.1



**Fig. 3.1:** Block scheme of the general program logic.

## 3.2 Chose of the technologies

When the general logic of the program is outlined, the choice of technologies for the implementation of the functionality can be proceeded.

### 3.2.1 Activity tracking

The pipeline for solving this problem includes block named "Body position" and has a dependency on "Bounding boxes" block, so the first task to solve is Human Pose Estimation (HPE).

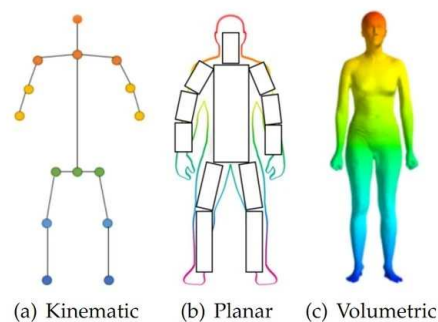
#### 3.2.1.1 Determination of the position of the human body

Since this project is not about learning about the HPE issue, but about using HPE to empower the home assistant, the logical solution would be to use off-the-shelf technology to solve the HPE problem.

To date, there are many ready-made technical solutions to this problem, both classical and using deep neural networks technology.

With the rapid development of deep learning solutions in recent years, deep learning has been shown to outperform classical computer vision methods in various tasks, including image segmentation or object detection. Therefore, deep learning techniques have brought significant advances and performance gains in pose estimation tasks. Based on this, within the framework of this project, when choosing a technology for determining the position of the human body, the focus will be on deep neural network technologies.

Majority of the deep learning based technical solutions for determining the position of the human body work on a similar principle. The input data is images or videos on which neural networks recognize elements of the human body. After that, using various methods, neural networks determine the position of the key points of the body. Most often, joints (elbows, knees, shoulders), large parts of the body (chest, torso, pelvis) and parts of the face (nose, ears, eyes) act as such key points. Based on these key points, a 2D or 3D model of the human body is built. In the case of a 2D model, this can be a skeletal model or a silhouette, in the case of a 3D model, this is a three-dimensional volumetric model of the human body (fig 3.2).



**Fig. 3.2:** Human pose models.

For this project, the most suitable output format would be a skeletal model - key points of the body corresponding to joints and connections between them corresponding to body parts.

HPE algorithms, according to the approach to determination of the position of the key points, are divided into two large groups: bottom-up and top-down approaches.

With top-down approach all the persons in a frame are detected first, and then regions of interest are created around them. After that, within these regions, the positions of key points are determined for each person and body models are built. More the number of persons, the more the computational complexity of the task. This approach is scale invariant. It performs well on popular benchmarks in terms of accuracy. However, due to the complexity of these models, achieving real-time inference is computationally expensive.

In the bottom-up approach, the algorithm first finds all the key points of all people in the image without identification, and then groups them depending on who these key points belong to. A probabilistic map called heatmap is used by this approach to estimate the probability of every pixel containing a particular key point. With the help of

Non-Maximum Suppression, the best landmark is filtered. This method is less complex compared to Top-down methods but at the cost of reduced accuracy.

The top-down approach is generally easier to implement because implementing a person detector is simpler than implementing associating or grouping algorithms. However, it is not easy to judge which approach can provide better performance. Overall performance depends on which works better: the associating or grouping algorithms or the person detector.

Taking into account the fact that within the framework of the project the program will work with only one person in the frame, it will be more profitable to use the bottom-up approach. However, with a high resolution input video stream, the top-down approach can be computationally more efficient, since the key point detection algorithm does not have to process the entire image.

In the course of work on this section, a study of tools and ready-made solutions that allow solving the problem of determining the position of the human body was made. The most popular and advanced solutions today are OpenPose, DeepCut, AlphaPose, HRNet, Deep Pose, PoseNet, DensePose, YOLO Pose and BlazePose.

These tools were compared in terms of input data requirement, speed, format of output data, terms of use and ease of implementation. Of these, the two most suitable for solving the problems of this project are BlazePose (Google Pose) and YOLO Pose. These solutions were chosen as one of the most modern, widely used and providing a user-friendly API. In addition, both of these solutions have one important feature for the project - they allow not only to determine the position of the human body, but also to determine the position of the hands. This is an important advantage, as it will allow solving two project tasks using one technology.

### **Mediapipe and YOLO comparison**

To ensure optimal accuracy, both MediaPipe and YOLOv7 require a significant amount of training data. MediaPipe provides pre-trained models that are trained on large datasets such as COCO, MPII, and OpenPose, ensuring high accuracy in a wide range of conditions. YOLOv7 requires users to manually collect and label training data, which can be a time-consuming and error-prone process. However, YOLOv7's modular architecture allows users to customize the model for their specific application and improve accuracy over time.

It should be noted that there are pre-trained YOLO Pose models in the public domain, however, most of them cannot be compared in terms of the size of the training dataset with the solution from Mediapipe. Even on the official page of the YOLO Pose repository, it is proposed to train the neural network on its own and noted that the provided pre-training models may show results that differ from the description [10].

Let's conduct a practical comparison of the work of YOLOv7 and Google Pose and choose the most appropriate technology.

Google Pose and Yolo Pose differ in the algorithm for determining the position of key points on the human body. YOLO, due to the way it works, detects the position of the human body on every frame, while Google Pose performs the detection procedure as needed and then switches to tracking mode. This approach gives a more stable result and reduces the frame processing time in tracking mode.

Due to the fact that Google Pose uses a combination of detection and tracking of key points of the body, while YOLO Pose performs a detection procedure for each frame, Google Pose has a significant FPS gain, which is an advantage for a system working with a real-time video stream.

Also, thanks to Google Pose key point tracking technology, it provides smoother movement of the key points, which significantly decreases their jitter. With YOLO Pose, due to the fact that it detects key points in each frame separately, the positions of the key points may slightly change from frame to frame. This feature can play an important role in the implementation of the functionality of interaction with smart home devices. Less jitter will allow to more accurately point to objects.

In addition, the tracking technology in Google Pose allows this solution to do a much better job of determining body position in cases where a person is far from the camera and his silhouette is small. This will allow the system to work in large rooms where a person can move away from the camera for a considerable distance.

YOLO Pose does a better job of recognizing partially occluded key points, however, within the scope of the project, this problem should not be significant as the cameras will be deliberately positioned in such a way as to provide the best view of the person.

YOLO Pose also shows slightly better results in low light conditions, but the difference is negligible. In addition, an infrared night camera can be used to solve the problems associated with working in low light and dark conditions.

When recognizing unusual postures, such as lying down, inverted body position, Google Pose shows a much better and stable result.

Considering that one of the main functions of this virtual assistant is fall detection, Google Pose can provide better input data for the algorithm responsible for recognizing the unnatural body position.

In terms of the number of tracked persons in one frame, YOLO Pose surpasses Google Pose, since YOLO Pose allows you to track an unlimited number of people in the frame, while Google Pose can only work with one person. However, given that this virtual assistant is being developed for elderly people living separately, this distinction does not play great role in the framework of the project.

The next difference between YOLO Pose and Google Pose is the number of key body points that the model defines. YOLO Pose has 17 key points, Google Pose has 33. Although most of the functionality of the virtual assistant can be implemented using the 17 key point model, for detecting unnatural body positions, more body key points can provide a more consistent result.

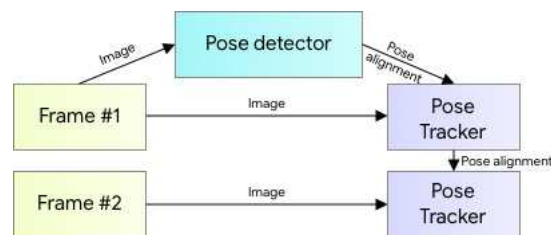
Another difference between these technologies is optimization. YOLOv7 is not optimized to work with CPU and edge devices, but its flexible architecture allows you to do it manually. Google Pose has built-in optimizations for CPU and edge devices. Since this project is a Proof of Concept, a more appropriate solution would be the one that works well on a variety of devices without the additional effort for fine-tuning and optimization [15].

Based on the above, Google Pose looks like the most appropriate technology to use in this project for solving the human pose estimation task.

### Google Pose operating principle

Google Pose is one of the modules of the Google Mediapipe framework. This technology is based on the BlazePose human pose estimation model from Google.

BlazePose consists of two machine learning models: a Detector and an Estimator (fig. 3.3). The Detector cuts out the human region from the input image, while the Estimator takes a 256x256 resolution image of the detected person as input and outputs the key points.



**Fig. 3.3:** Human pose estimation pipeline overview.

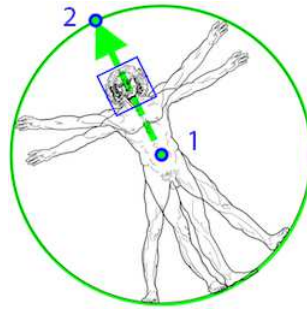
The Detector is an Single-Shot Detector(SSD) based architecture. Given an input image, it outputs a bounding box and a confidence score. There are two ways to use the Detector. In box mode, the bounding box is determined from its position (x,y) and size (w,h). In alignment mode, the scale and angle are determined from two key points that firmly describe the midpoint of a person's hips (1), the radius of a circle circumscribing the whole person (2), and the incline angle of the line connecting the shoulder and hip midpoints (fig. 3.4). In such a way the bounding box including rotation can be predicted.

The Estimator uses heatmap for training, but computes key points directly without using heatmap for faster inference (fig. 3.5).

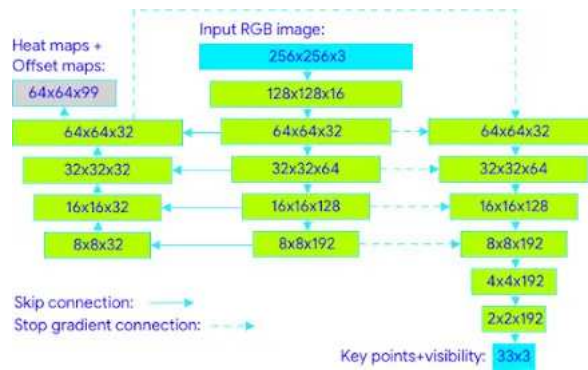
The output of the Estimator is landmarks. The landmarks are made of 165 elements for the (x, y, z, visibility, presence) for all 33 key points.

The z-values are based on the person's hips, with key points being between the hips and the camera when the value is negative, and behind the hips when the value is positive.

The visibility returns the probability of key points that exist in the frame and are not occluded by other objects. presence returns the probability of key points that exist in the frame.

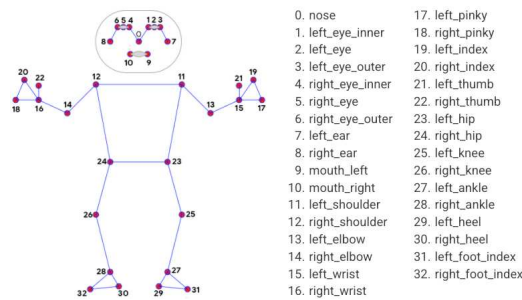


**Fig. 3.4:** Vitruvian man aligned via two virtual keypoints predicted by BlazePose detector in addition to the face bounding box.



**Fig. 3.5:** Tracking network architecture: regression with heatmap supervision.

BlazePose outputs the 33 key points according to the ordering convention shown on fig. 3.6. This is more points than the commonly used 17 key points of the COCO dataset [16].



**Fig. 3.6:** MsediaPipe Pose landmarks.

### 3.2.1.2 Determination of user location

According to the coordinates of the key points of the user’s body, it will be determined in which part of the camera’s field of view he is. In order to determine which part of the room the user is located, the camera’s field of view will be divided into areas representing different parts of the room: "kitchen", "workplace", "bed", "dining table", etc. All movements between these zones will be logged to a separate file for further processing.



To delimit the camera's field of view into separate areas, a user graphical interface that allows to manually create these areas in the frame, delete or change them, will be used. Information about these areas will be saved in a separate file so that after a system restart they will be loaded automatically. This functionality will be provided by the "Bounding boxes" block.

The user's presence in a specific area will be determined by analyzing the position of their body within the camera's field of view, relative to predefined zones. This is achieved by comparing the coordinates of the user's body with those of the designated areas.

### 3.2.2 Fall detection

The fall detection pipeline consists of two blocks: "Body position" and "Fall detection" itself. Also, this solution has dependency on "Bounding boxes" block.

The task can be decomposed into two subtasks according to these blocks:

1. Processing of the input video stream in order to determine the position of the user's body.
2. Classification of the unnatural pose based on data about the position of the user's body.

Thus, the solution of the problem of determining the unnatural position of the body will consist of two stages.

#### 3.2.2.1 Determination of the position of the human body

To determinate the position of user body, the information obtained at the output of the "Body position" module will be used. The choice of technology for its implementation and the principle of its operation were presented in the previous subsection.

#### 3.2.2.2 Fall detection

The task of determining the unnatural position of the user on the video can be solved in several ways. One method is to classify the user's state or activity by the method of direct processing of the images as described in [12] [13] [14].

This method allows to determine a fairly wide range of human actions in the frame, but for this project this is not the best solution. The disadvantage of this method is that the output is only a classification of human actions and nothing more. At the same time, to implement the function of collecting data about the user's activity, information about his position in the frame is needed, and to implement the function of controlling smart home devices, data on the position of the user's body parts relative to each other is necessary. So, instead of this method, human pose estimation algorithm is used.

This solution has several advantages over the method described above. Firstly, data on the position of key points of the body in the frame can be reused to solve the problems of collecting information about user activity and managing smart home devices. Thus, one procedure will provide two more modules with input data, which will greatly simplify the program and speed up its work as the computationally expensive task of processing the video stream will be solved once for multiple modules. Secondly, such an approach greatly simplifies the technical solution of the problem of classifying an unnatural body position. Since the input data for the neural network that classifies the unnatural pose will be in the form of key points and their coordinates in the frame, the process of preparing training data and training the neural network will be faster, and the architecture of the neural network will be much simpler. Also, it is much easier to train a neural network on a set of numerical data than on a set of images.

The task of classifying the unnatural position of the user can be solved using different approaches. Data on the position of key points allow using classical methods, such as creating a set of instructions for calculating such parameters as distances between key points and angles between limbs and body parts. Next, a set of conditions can be created for these parameters, under which the pose will be classified as unnatural. This approach is described in one of the examples of the Google Pose documentation [17].

A serious problem with this approach is the large variability in the relative position of the key points of the user's body. This entails excessive complexity of the algorithms for calculating the parameters mentioned above and the need to create very complex conditions.

Another approach to solving the problem of determining the unnatural position of the body is to use deep neural networks. The main advantage of neural networks lies in their ability to learn from experience, without the need for explicit programming for each specific task. They are able to work with large amounts of data and highlight complex and non-obvious patterns that cannot be identified using traditional data analysis methods.

Given the above, the neural network looks like the most appropriate tool for classifying the unnatural body position based on data on the relative position of its key points. Such a solution will save us from the need to develop complex instructions and conditions, which will significantly reduce labor costs for implementation.

At the moment, there are many frameworks for creating and training neural networks. Some of the most popular and widely used are TensorFlow, PyTorch, TensorFlow and Keras. The choice of a framework for creating a neural network depends on several factors, such as the level of experience of the developer, the availability of computing resources, and the type of problem being solved.

To solve the problem of classifying an unnatural position, we need a network with 99 neurons in the input layer for 33 key points, each of which has three coordinates. The output layer will consist of one neuron, since the neural network solves the problem of binary classification. By the standards of modern architectures, this is a rather small

neural network for which a simple framework is suitable. Keras is well suited for creating simple models and prototypes. In addition, it has a fairly simple and intuitive API and is optimized to work with the CPU. Keras is an add-on to the TensorFlow library that solves machine learning problems. TensorFlow performs all low-level calculations and transformations and serves as a mathematical core. Keras, on the other hand, serves as a high-level API to simplify the process of building and training models.

As it was said in chapter 1, not all cases in which a person is in a horizontal position are indicative of a critical situation. For example, when the user is lying on a bed, such a position should not be classified as an unnatural pose. To prevent false triggering of the unnatural pose classification, some zones in the camera's field of view will be marked as "safe areas". In these areas the unnatural pose classification will be suppressed. The ability to create these areas will be provided by the "Bounding boxes" block using the functionality of the OpenCV library.

### 3.2.2.3 Summary

Let's summarize the choice of technologies for solving the problem of determining the unnatural position of the user's body.

This task was decomposed into two subtasks:

1. Defining the position of the user's body in the form of body key points, each of which has three spatial coordinates.
2. Classification of the unnatural position of the body based on the coordinates of its key points.

This division was made to reuse the data obtained at the output of the first subtask for the implementation of the rest of the project functionality.

The first subtask will be solved using the Google Pose tool from the Google Mediapipe framework. This technology was chosen due to the presence of a high-quality pre-trained model for determining the position of the human body, ease of use, and the presence in the Google Mediapipe framework a tool for determining the position of the hands, which can be used to solve the problem of controlling smart home devices.

The second subtask will be solved using deep learning technology. The choice is due to the fact that in order to classify the unnatural position of the body, it is necessary to identify rather complex and non-obvious patterns in the relative position of the key points of the body. The solution of this problem by classical methods will require rather large labor costs and a large amount of code, while for a neural network this procedure is a reference task. Due to the fact that the project is a proof of concept, and the neural network that solves the problem of classifying an unnatural pose will be small, Keras as a well suited for prototyping, simple framework with a high-level API, was chosen.

To manage the video stream, and creating bounding boxes around safe areas the OpenCV library will be used.

### 3.2.3 Smart home device control

The device control pipeline includes three blocks: "Hands position", "Hand gestures classification" and "Device control", which are necessary to classify a command from gesture and transmit it to a device.

Also, this pipeline has dependency on such blocks as "Bounding boxes" and "Body position". These blocks provide functionality for device selection.

To implement the functionality of controlling smart home devices using gestures, it is necessary to solve four tasks:

1. Select a device for interaction
2. Determine the position of user's hands
3. Classify a gesture based on hand position data
4. Send a command to a device

#### 3.2.3.1 Selecting devices for interaction

In voice home assistants that combine smart home devices into one system, the process of choosing a device to interact with is quite simple. Due to the fact that modern technical solutions based on natural language processing are able to perceive rather complex commands and language constructs. In case of CV-based virtual assistant this process is little more complicated. As mentioned in chapter 2, one of the easiest and most natural way to select a smart home device is to point to it with a hand.

To implement this function, it is necessary to have data on the position of the user's arms and on the location of the device in the camera's field of view.

The task of determining the position of the user's arms has already been partially solved by the "Body position block". To determine the position of the arms, the corresponded key points can be selected from the list of key points of the body. Based on the coordinates of these key points, pointing beams can be built, with the help of which it will be possible to select a device for interaction.

In order for the device to be selected using the pointing beam, the program must have information about its position in the frame. As outlined in the functional requirements section, each device in the frame will be marked by a bounding box. In such a way the program will be provided with the information about device location in the frame. The device will be considered selected when the pointing beam and the corresponding bounding box intersect.

The functionality that allows to create bounding boxes around devices, as well as automatically save them when the program ends and load them when it starts, is implemented by the "Bounding boxes" block.

### 3.2.3.2 Determining the position of the hands

As mentioned in the part about choosing a technology for determining the position of the human body, the Google Mediapipe framework includes a tool for determining the position of the hands.

To reduce the amount of code and used libraries, a reasonable solution would be to use the Google Mediapipe functionality to solve several project tasks. Thus, to solve the problem of determining the position of the hands, the Google Hands tool will be used.

#### MediaPipe Hands operating principle

MediaPipe Hands is a hand and finger tracking solution. It utilizes an ML pipeline consisting of multiple models working together: a palm detection model that operates on the full image and returns an oriented hand bounding box and a hand landmark model that operates on the cropped image region defined by the palm detector and returns high-fidelity 3D hand keypoints.

Providing the accurately cropped hand image to the hand landmark model drastically reduces the need for data augmentation (e.g. rotations, translation and scale) and instead allows the network to dedicate most of its capacity towards coordinate prediction accuracy. In addition, in this pipeline the crops can also be generated based on the hand landmarks identified in the previous frame, and only when the landmark model could no longer identify hand presence is palm detection invoked to relocalize the hand.

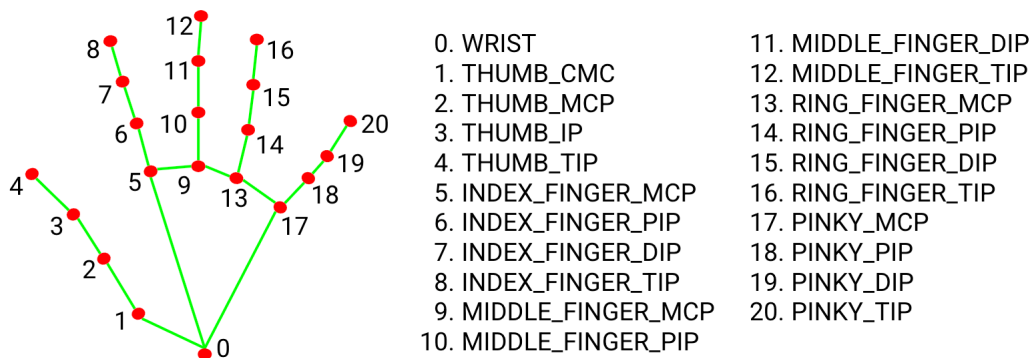
To detect initial hand locations, a single-shot detector model is used. The model is able to work across a variety of hand sizes with a large scale span ( 20x) relative to the image frame and can detect occluded and self-occluded hands.

The model solves hand detection task using different strategies. First, a palm detector is used instead of a hand detector, since estimating bounding boxes of rigid objects like palms and fists is significantly simpler than detecting hands with articulated fingers. In addition, as palms are smaller objects, the non-maximum suppression algorithm works well even for two-hand self-occlusion cases, like handshakes. Second, an encoder-decoder feature extractor is used for bigger scene context awareness even for small objects. Lastly, the focal loss during training was minimized to support a large amount of anchors resulting from the high scale variance.

With the above techniques, an average precision of 95.7% was achieved in palm detection. Using a regular cross entropy loss and no decoder gives a baseline of just 86.22

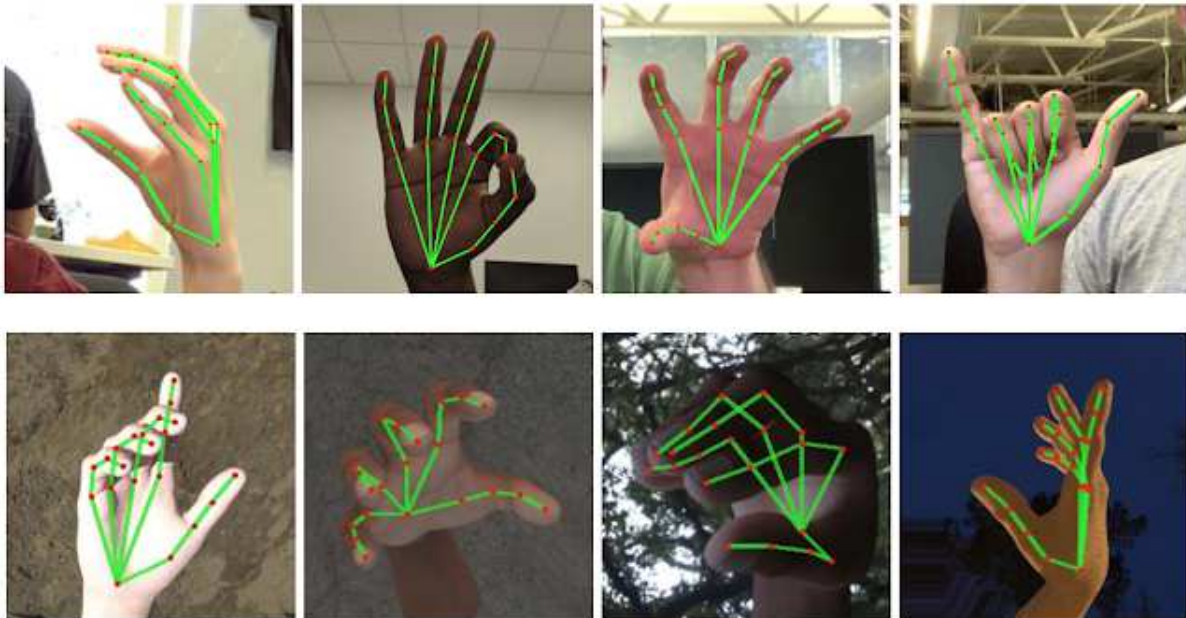
After the palm detection over the whole image a subsequent hand landmark model performs precise key point localization of 21 3D hand-knuckle coordinates inside the detected hand regions via regression, that is direct coordinate prediction. The order of the key points is represented on fig. 3.7. The model learns a consistent internal hand pose representation and is robust even to partially visible hands and self-occlusions.

To obtain ground truth data, 30K real-world images with 21 3D coordinates were manually annotated (fig. 3.8, first row). To better cover the possible hand poses and



**Fig. 3.7:** 21 hand landmarks.

provide additional supervision on the nature of hand geometry, a high-quality synthetic hand model over various backgrounds were rendered and mapped to the corresponding 3D coordinates (fig. 3.8, second row) [11].



**Fig. 3.8:** Aligned hand crops passed to the tracking network with ground truth annotation. Bottom: Rendered synthetic hand images with ground truth annotation.

### 3.2.3.3 Gesture command classification

The classification of the gesture will be based on the data on the position of the key points of the user's hands, obtained at the output of the "Hands position" block. The format of this data is similar to the format of data about the position of the user's body, and the task of classifying a gesture is also to find patterns in the relative position of key points. Since the task of determining the hand gesture and the task of determining the position of the human body are almost identical, the same set of technologies and methods can be used to solve them. Based on the foregoing, the task of classifying hand

gestures will be solved using deep learning technology, and the Keras framework will be used to create a model.

#### 3.2.3.4 Sending a command to a device

Based on the data received from the output of the "Hand gestures classification" block, a command for the selected device will be generated. The vast majority of modern smart home devices have a user API for various programming languages in the public domain. Interaction with the device through API is the simplest and most natural way, so the most logical solution would be to take advantage from it.

#### 3.2.3.5 Summary

Let's summarize the process of choosing technologies for implementing the functionality of smart home devices control.

Technical implementation was decomposed into following tasks:

1. Selection a device for interaction.
2. Determination Of the position of user's hands.
3. Classification of a gesture based on hand position data.
4. Sending a command to a device.

The choice of device for interaction will occur by pointing at it with hand. For this, data on the position of the key points of the user's body and bounding boxes around the devices will be used. Based on the position of the key points of the user's arms, a pointing beam will be built. The device will be selected when this beam crosses the bounding box corresponding to the device.

To select a command for the device, gesture signs will be used. To do this, first the position of the user's hands will be determined by the MediaPipe Hands tool. Data on the position of the key points of the hands will be transferred to the input of the neural network, which classifies the gestures. Based on how the neural network classifies the gesture, a command for the device will be selected. Communication between the virtual assistant and the device will be implemented through the API provided by the device manufacturer.

### 3.2.4 Camera requirements

After the set of technologies that solve the main tasks of the project was stated, the requirements for the input data source will draw up.

As said above, the input data will be a video stream from one or more cameras in real time. The full-featured version of the product will use multiple cameras. This is necessary

to solve two technical problems. Firstly, two cameras will provide the effect of binocular vision, which will allow to more accurately determine the position of the user's body in all three dimensions. Secondly, in order to collect user activity statistics, the cameras must cover all walk-through rooms and rooms where the user spends most of the time. However, this project is proof of concept and its main task is to study the system's performance and determine the possibility of its applications, therefore, within the framework of this work, the input video stream will come from only one camera.

Camera requirements will include two main parameters: resolution and viewing angle. These two parameters are dependent on each other. A larger camera angle allows to cover more space of the room in the field of view. However, with an increase in the viewing angle at the same resolution, the detail of objects decreases, since a larger area of space is projected onto the same area of the camera sensor.

So, the combination of the viewing angle and the resolution should be chosen so that the silhouette of a person in the frame at the maximum possible distance has sufficient resolution to determine the position of the body and hands.

It was experimentally determined that for a stable determination of the position of the body, it is enough that the silhouette of a person has about 100 pixels in height. To determine the position of the hands, the resolution requirements are much higher. In order for the neural network to determine the positions of the joints of the hand, fingers must be distinguishable. To do this, the hand on the image must have a resolution of at least 30\*30 px.

From the foregoing, it follows that the camera's viewing angle should be chosen as small as possible to cover the space in which the user will be located. As for the video resolution, it chosen in such a way to provide necessary level of detalization.

Since the project is a proof of concept, during development and experimentation, a laptop web camera will be used as the easiest option to implement.



# 4

## Technical implementation

Once the technologies and methods necessary for resolving the project's challenges have been identified, the next step is to proceed with its technical implementation.

For the convenience of developing and supporting the project, its software part will be divided into separate modules depending on the tasks performed. Modules will be linked by dependencies to be able to call each other's functions. Initialization and launch of individual modules will be carried out from the main program file. The diagram at fig 4.1 shows program structure.

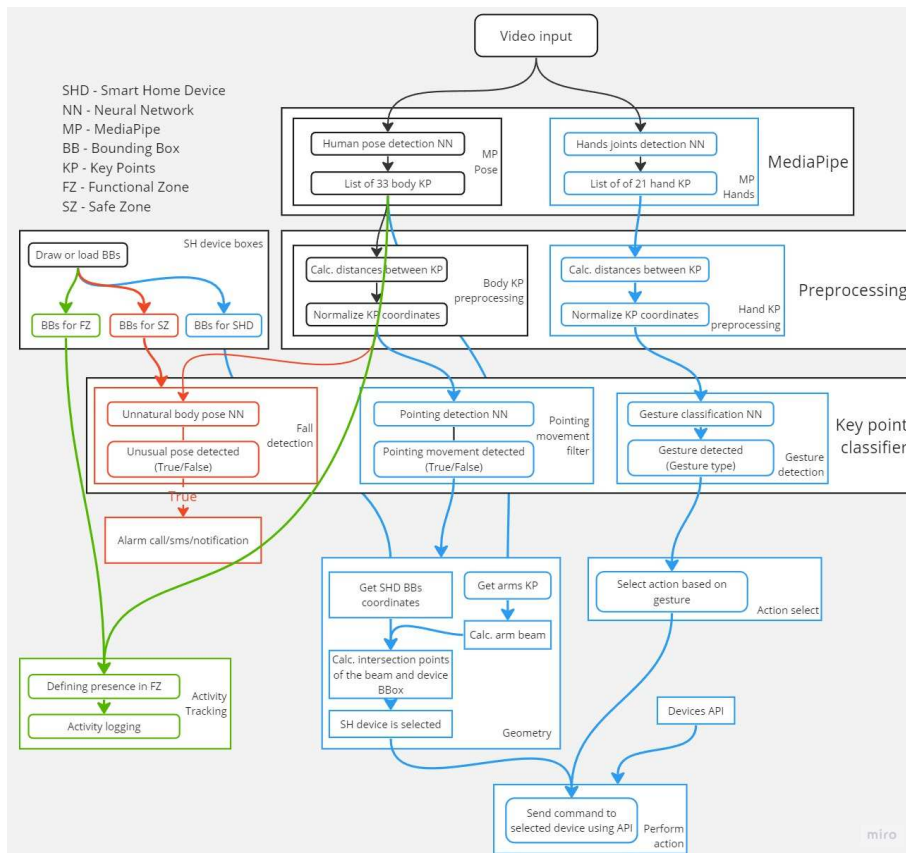


Fig. 4.1: Block scheme of the virtual smart home assistant.

## 4.1 Receiving video input

The initial step is to receive a video signal and preprocess it for transmission to subsequent modules. OpenCV provides a wide range of functionalities for real-time video stream processing, including operations like cropping, color model manipulation, resolution changes, and more. To facilitate the handling of video input streams and displaying the results, we will create a module named *frame\_handler.py*.

Firstly, a function to initialize OpenCV and receive the input video stream will be defined. The initialization function allows for selecting the desired input video source and creating a window to display the output video.

```
1 # Function to init cv2 video input
2 def init_cv2(input_source):
3     global cam
4     cam = cv.VideoCapture(input_source)
5     cv.namedWindow('Video stream')
```

The *start\_cv2()* function begins a loop that iterates through the input video stream, processing and displaying the video in a window. This loop serves as the core for all frame-by-frame operations involved in both input and output video processing.

```
1 # Function that starts cv2 loop
2 def start_cv2():
3     while cam.isOpened():
4         cam_ret, frame = cam.read()
5         if input_key == 27: # ESC
6             break
7         cv.imshow('Video stream', frame)
```

The result of these actions is the ability to capture video from the camera and display it in real time. After the input data stream was setted up, it can be passed to subsequent blocks.

## 4.2 Activity tracking

Once the input video is acquired, the next step involves the task of gathering statistical information pertaining to user activity. The core principle revolves around tracking and logging the user movements between various rooms or specific areas within a room throughout the day.

In scenarios where multiple cameras are utilized across different rooms, the user's location can be determined by the camera whose field of view captures the presence of the user.

To achieve a more precise user location within the field of view of a single camera, its frame will be divided into zones that correspond to the functional areas of the room.

By dividing the frame into these zones, the system can provide more detailed information about which specific area within the room the user is located.

In order to solve this problem it is necessary:

1. Divide camera frame into zones using bounding boxes;
2. Determine the position of the user in the frame;
3. Based on the data on the position of the user in the frame and on the functional zones' bounding boxes, determine in which room or part of the room he is located;
4. Keep track of the time spent by the user in the room or part of the room;
5. Save the collected data;

### 4.2.1 Divide camera frame into zones

The OpenCV library provides functionality to draw geometric shapes on the frame and capture mouse clicks' coordinates. Leveraging this capability, a simple functionality has been implemented to draw rectangles or other shapes on the video output window, indicating safe zones. As this functionality will also be useful for designating smart home devices and excluded zones of the subsequent tasks, it is advisable to move it to a separate module called *bboxes.py*. This modularization will ensure a more organized and maintainable code structure.

### 4.2.2 Activity tracking

As previously mentioned, the recognition of individuals within the frame and the identification of key body landmarks will be accomplished using the Google Pose tool. To streamline the process, the recognition of individuals, extraction of key body points, and any other related operations will be grouped in the "*pose\_processing.py*" module. This module will handle the processing and analysis of the pose data obtained from the input video.

A dedicated function has been implemented to initialize Google Pose, offering the flexibility to customize its settings at program startup.

```
1 # Function to init MediaPipe Pose
2 import mediapipe as mp
3
4 def init_mp_pose(detect_conf, track_conf, static_img, complexity):
5     global pose_solution, pose_detector
6     pose_solution = mp.solutions.pose
7     pose_detector = pose_solution.Pose(
8         static_image_mode=static_img,
9         model_complexity=complexity,
10        min_detection_confidence=detect_conf,
```

```

11     min_tracking_confidence=track_conf
12 )

```

The *static\_image\_mode* parameter defines the input data format. Setting it to True indicates a static image, while setting it to False indicates a video stream. The *model\_complexity* parameter allows to choose the size of the model used to determine the body's key points. A more complex model enhances accuracy but may impact system speed. The *min\_detection\_confidence* parameter represents the minimum confidence threshold for successful person detection within the frame. Moreover, the *min\_tracking\_confidence* parameter sets the minimum confidence level for tracking the key points. If the model's confidence in tracking falls below this value, the person detection procedure is triggered in the subsequent frame. Once successful detection occurs, the system switches back to tracking mode.

After initialization, processing of the input video stream can be started using Google Pose. Since the Google Mediapipe models work with images in the RGB color model, and the OpenCV library in the BGR model, it is necessary to convert the color models before passing the frame to the Google Pose input.

After that, the frame can be passed to the *get\_pose\_landmarks()* method, which, after processing the image, will return a set of body key points with their coordinates. The Mediapipe API provides a function to display key points on an image. To control the operation of the system, let's superimpose them on the frame of the output video, after converting the color model of the frame back to BGR.

```

1 # Process image for pose
2 frame = cv.cvtColor(frame, cv.COLOR_BGR2RGB)
3 frame.flags.writeable = False
4 pose_landmarks = pose_proc.get_pose_landmarks(frame)
5 frame.flags.writeable = True
6 frame = cv.cvtColor(frame, cv.COLOR_RGB2BGR)

```

The result of the performed operations is presented on the fig. 4.2. In the program, a set of key points is represented as a list and has the following form:

```

1 [x: 0.29373878240585327
2  y: 0.6394885778427124
3  z: -0.6754010915756226
4  visibility: 0.9958284497261047
5
6  x: 0.3330877721309662
7  y: 0.5740934610366821
8  z: -0.6887555718421936
9  visibility: 0.9957334399223328
10
11 ...
12
13 x: 0.24768665432929993

```

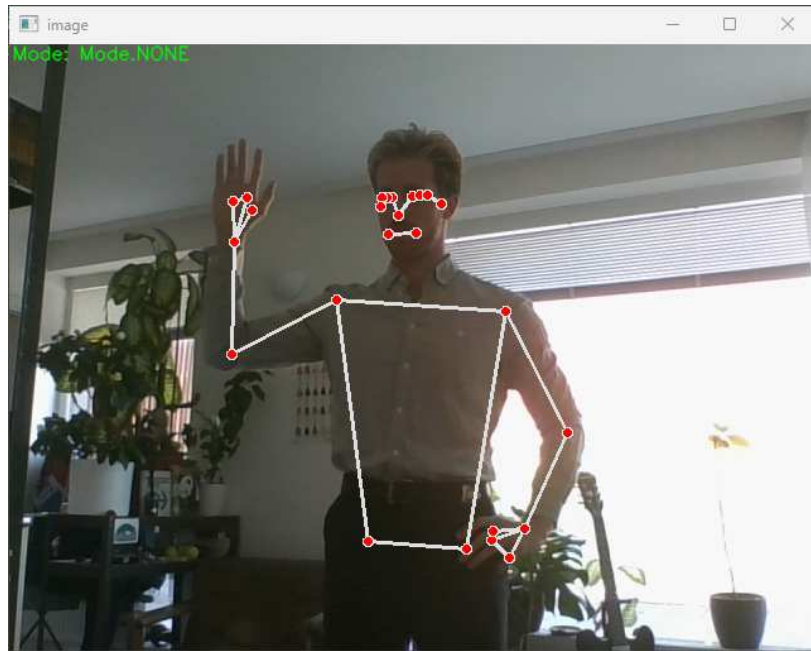


Fig. 4.2: Body key points superimposed on video stream.

```
14 y 2.9935364723205566  
15 z: 0.6450490951538086  
16 visibility: 0.000280229403]
```

#### 4.2.2.1 Determination of user location

To divide the camera frame into zones, the *bboxes.py* module will be used, which allows drawing bounding boxes.

Information pertaining to the user's movement throughout the home, such as entering or leaving different areas, will be logged along with corresponding timestamps. These recorded data will enable the calculation of the amount of time the user spent in each functional area and can be used for further analysis.

The user's presence in a specific zone will be determined by the simultaneous positioning of key points 23 and 24, which correspond to the hip joints, within the designated area. These points have been selected due to their proximity to the center of mass of the human body, making them optimal for accurately reflecting the person's spatial position. Other key points have been excluded from consideration as they possess greater mobility in relation to the center of mass and may exit the designated areas while the user stays in it.

Data about the user's movement around the premises will be regularly saved to a dedicated file in the form of a log.

## 4.3 Fall detection

The Fall Detection task continuously monitors the position of the user's body and classifies whether their pose is normal or unnatural. To ensure accurate detection and prevent false alarms, certain areas have been designated where an unnatural pose should not be classified. To address this issue, the following steps need to be taken:

1. Determine the position of the user;
2. Classify user unnatural pose;
3. Define excluded zones within the frame;
4. Report or ignore unnatural pose based on information about user location and excluded zones;

### 4.3.1 Determining the position of the user's body

The functionality for solving the first the third and the fourth subtasks has already been implemented when solving the problem of user activity tracking. To determine the position of the user in the frame, the key points of the his body will be used. To divide the camera frame into zones, the *bboxes.py* module will be used, which allows drawing bounding boxes.

### 4.3.2 Fall detection

#### 4.3.2.1 Pose classification

To address the problem of recognizing unnatural positions based on the key points of the user's body, deep neural networks can be employed. In this case, the Keras framework is chosen as it offers a high-level API for creating and training neural networks.

Here is a step-by-step approach to using deep learning technology for solving the problem:

1. Data collection: The first step is to collect a labeled set of data about the positions of key points of the body, along with corresponding labels indicating whether they represent natural or unnatural positions.
2. Data preprocessing: The collected data will be prepared for training. This step involves tasks such as normalizing the input key point positions and splitting the dataset into training and validation set for training the neural network.
3. Model architecture design: The architecture of the neural network will be designed using Keras' API. Depending on the nature of the data and the problem at hand, it will be considered which deep learning models to use. Experiments with different

architectures and layer configurations will be provided to identify the most suitable model for the task.

4. Model compilation: The model will be compiled by specifying the loss function, optimizer and other metrics which will be tracked during training.
5. Model training: The model will be trained using the preprocessed dataset. Keras' `fit()` function will be used to train the model on the training data, providing the number of epochs, batch size, and validation data. The training progress will be monitored by observing the loss and any other metrics specified during compilation.
6. Model evaluation: The trained model will be evaluated on the validation set to assess its performance. This step helps to understand how well the model generalizes to unseen data. Metrics such as accuracy, precision, recall, and F1 score can be used to evaluate the model's performance.
7. Model optimization: If the model's performance is not satisfactory, experiments with different hyperparameters, model architectures, or data augmentation techniques will be repeated to improve its effectiveness.
8. Model deployment: When the performance of the model is satisfactory, it will be saved for future use and deployment to the project environment.

To develop an effective and accurate model, these steps will be iterated , making adjustments to parameters and configurations as necessary.

### Data Collection

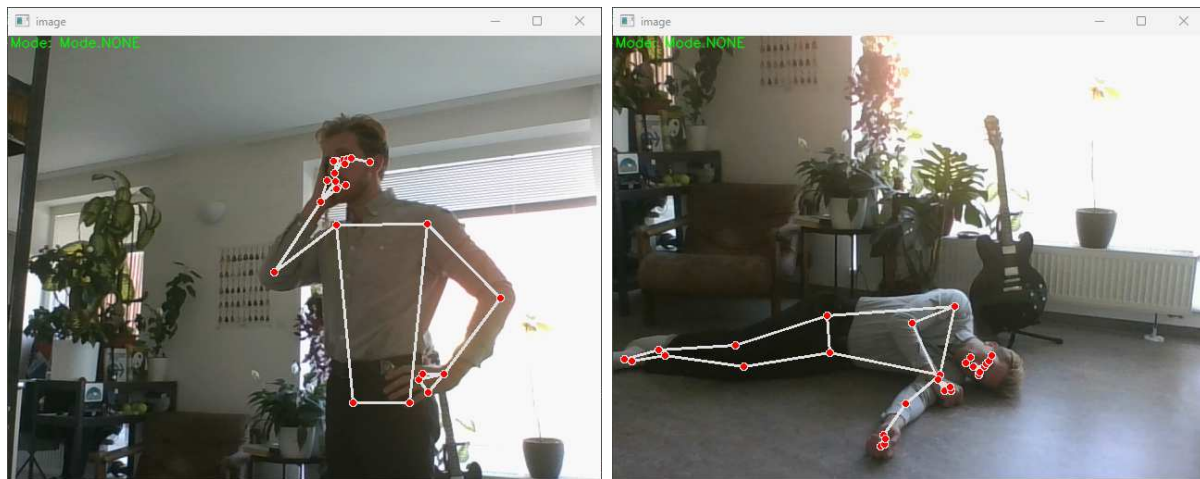
In order to proceed with experimenting on the neural network architecture, it is necessary to prepare a dataset for training. An auxiliary program has been developed, separate from the virtual assistant, to facilitate this task. This program incorporates same body position tracking functionality as described earlier and provides an ability to save a set of key point coordinates into a separate file in the .csv format by pressing a designated key on the keyboard. The key point coordinates will be saved in the following format:

```
0, x0, y0, z0, x1, y1, z1, ..., x32, y32, z32,  
1, x0, y0, z0, x1, y1, z1, ..., x32, y32, z32.
```

In the provided format, each line of the .csv file represents a single instance or sample in the dataset. The first value in each line(0, 1), serves as an label for the sample, where "0" indicates a normal pose, and "1" indicates an unnatural pose. The specific value of the label will depend on the key that was pressed during the key point tracking process. The subsequent values represent the coordinates of the key points, where (x0, y0, z0) corresponds to the coordinates of the first key point, (x1, y1, z1) corresponds to the coordinates of the second key point, and so on until the last key point (x32, y32, z32).

By saving the key point coordinates in this standardized format, these data can easily load the into a deep learning framework, such as Keras, for further preprocessing, model training, and evaluation. This format ensures the consistency and compatibility of the dataset with the deep neural network architecture.

The process of collecting data for training is demonstrated on fig. 4.3. The left image is an example of a normal body position in which key points are saved with label "0". The right image represents the unnatural pose that was saved with label "1".



(a) Normal pose.

(b) Unnatural pose.

**Fig. 4.3:** Examples of normal and unnatural pose.

To train a neural network for binary classification, a balanced dataset consisting of 400 elements was assembled, with 200 instances for each label. This balanced dataset ensures an equal representation of both classes, contributing to fair and unbiased training of the model.

Having an equal number of instances for each label helps prevent the neural network from being skewed towards one class during training. This approach enables the model to effectively learn the distinguishing features and patterns associated with each label. This will enable the model to classify new instances into the correct binary categories with better accuracy and reliability.

### Data preprocessing

Before proceeding with the creation and training of the machine learning model, it is crucial to address a problem related to the coordinates of the key points. The key point coordinates, calculated using the Google Pose tool, are relative to the frame and can vary depending on the person's position within the frame. This dependence of key point coordinates on the person's position can be captured by the neural network during the learning process, which will adversely affect the neural network's performance and degrade the quality of classification.



To mitigate this issue, it is necessary to normalize the key point coordinates. Normalization is a process that transforms the coordinate values to a standardized range, eliminating the influence of the person's position in the frame. By normalizing the coordinates, their relative scale and position become consistent across different instances, allowing the neural network to focus on the underlying patterns rather than the specific location within the frame. Normalization ensures that the key point coordinates have a comparable scale and are not biased by their position. This enhances the model's ability to learn the relevant features of the poses and improves the accuracy and generalization capabilities of the classification.

In this normalization approach, the coordinates of the key points will be transformed relative to a specific key point, denoted as index 0.

The normalization process involves replacing the original key point coordinates with their distances from the key point at index 0 along each axis. This transformation ensures that the model focuses on the relative positions of the key points rather than their absolute coordinates in the frame.

```

1 # Pose keypoint preprocessing
2 def pose_preprocess_to_base_point_xy(landmark_list):
3     temp_landmark_list = copy.deepcopy(landmark_list)
4
5     base_x, base_y, base_z = 0, 0, 0
6
7     for index, landmark_point in enumerate(temp_landmark_list):
8         if index == 0:
9             # set landmark 0 as base point
10            base_x, base_y = landmark_point[0], landmark_point[1]
11
12            # for each other landmark calculate it's
13            # distance from base point
14            temp_landmark_list[index][0] = temp_landmark_list[index][0] - base_x
15            temp_landmark_list[index][1] = temp_landmark_list[index][1] - base_y
16            temp_landmark_list[index][2] = temp_landmark_list[index][2] - base_z
17
18            # flatten landmark list (make it 1D)
19            temp_landmark_list = list(
20                itertools.chain.from_iterable(temp_landmark_list))
21
22            # Normalize distances of the landmarks from base point
23            temp_landmark_list = normalize(temp_landmark_list)
24            return temp_landmark_list

```

Additionally, a final scale normalization step is performed to ensure that all values fall within the range of -1 to 1, inclusive. This is achieved by dividing all values by the maximum value obtained after the previous transformation.

```

1 # Scale normalization
2 def normalize(flatten_landmark_list):

```

```
3     # Find max value of the list for normalization
4     max_value = max(list(map(abs, flatten_landmark_list)))
5
6     normalized_landmark_list = []
7
8     # Divide each value by max value
9     for element in flatten_landmark_list:
10        normalized_landmark_list.append(element / max_value)
11
12    return normalized_landmark_list
```

To ensure consistent normalization of data both during model training and during the operation of the model in the project environment, it is advisable to the preprocessing functionality to from the auxiliary program to the project. For preprocessing tasks solving, a preprocessing module called "preprocessor.py" will be created in the project.

### Model architecture design

The first step in designing the model architecture is to select an appropriate type of neural network based on the task at hand. In the case of binary classification with one-dimensional vectors of floating-point values, where the dataset elements are time-independent and independent from each other, recurrent and convolutional mechanisms are not required.

An optimal approach is to begin with a simple feedforward neural network.

Based on the data obtained from the HPE (Human Pose Estimation) block, certain parameters can be already determined for the neural network architecture.

In this case, 33 key points will be passed as input to the neural network. Each key point is represented by three coordinates (x, y, z), resulting in a total of 99 input features. Therefore, the input layer of the neural network should consist of 99 neurons, each corresponding to one input feature.

Since determining the unnatural body position is a binary classification task, the output layer of the neural network will consist of one neuron. The activation state of this output neuron can represent the classification decision. For instance, an active state (1) of the output neuron could indicate that the pose has been classified as unnatural, while an inactive state (0) could correspond to a normal pose.

Next, the decision needs to be made regarding the number of inner layers in the neural network. A good starting point is to create a neural network with a single hidden layer and assess its performance. The impact of adding more layers can then be examined to determine how it affects the overall performance.

To determine the appropriate number of neurons in the inner layer, valuable insights can be found in the book "Introduction to Neural Networks with Java." [18]. There are three tips:

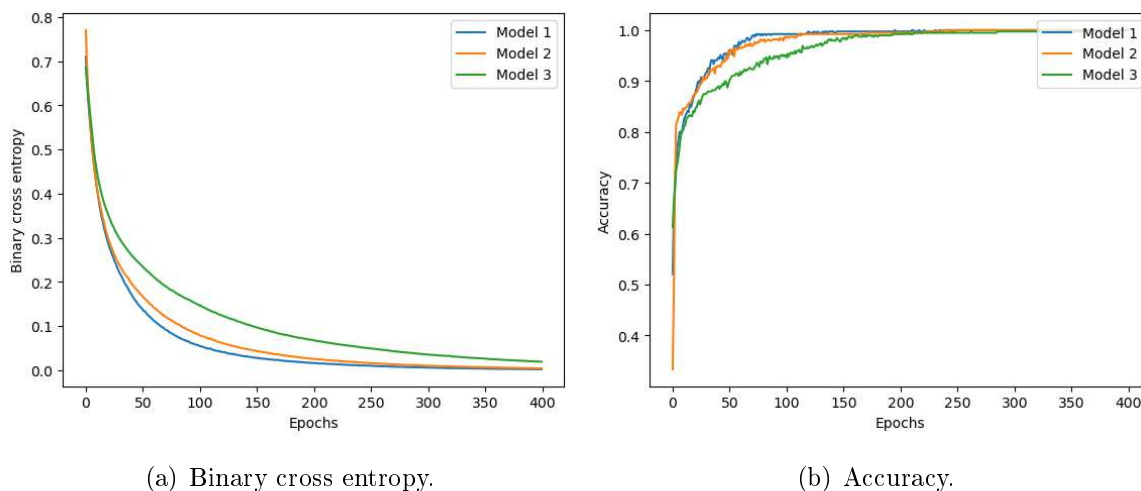
- The number of hidden neurons should be between the size of the input layer and the size of the output layer.

- The number of hidden neurons should be  $2/3$  the size of the input layer, plus the size of the output layer.
- The number of hidden neurons should be less than twice the size of the input layer.

Three models were created with varying numbers of neurons in the hidden layers. The first model consists of 99 neurons in the hidden layer, equivalent to the number of neurons in the input layer. The second model has 50 neurons, which is half the size of the input layer, and the third model contains 25 neurons, representing one-fourth of the input layer size.

In the test, the activation function "ReLU" was chosen for the layer between the input and the hidden layer. The input layer utilized the "sigmoid" activation function, commonly used in binary classification neural networks. The training loss function was set to "binary cross entropy", which is widely employed for this type of task. Additionally, accuracy, a common metric for evaluating performance, was plotted to obtain further insights into the performance of these neural networks.

The results of the initial test is represented on fig. 4.4.



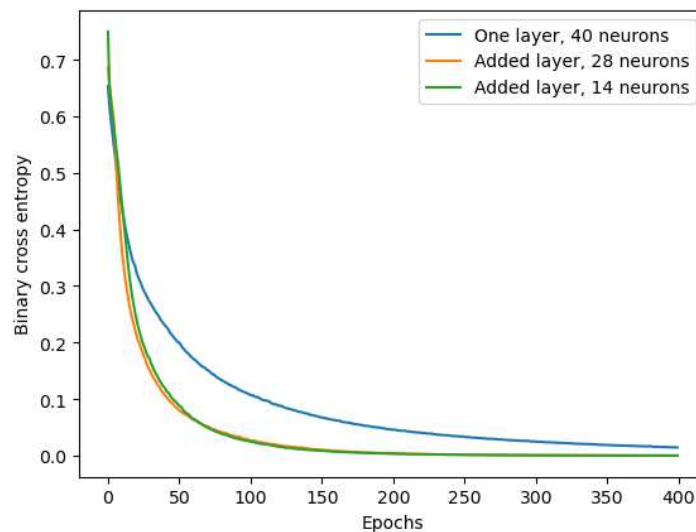
**Fig. 4.4:** Neural network comparison with varying numbers of neurons in the hidden layer.

As observed in fig. 4.4, the half-sized hidden layer perform equally well compared to the full-sized layer, while the neural network with the smallest hidden layer requires more epochs to achieve comparable results.

Further iterations of the experiment revealed that having 40 neurons in the hidden layer is sufficient to achieve performance comparable to the full-sized hidden layer within 400 epochs.

After determining the appropriate number of neurons for the first hidden layer, a second layer was introduced. Two configurations were experimented with: a layer consisting of  $2/3$  of the neurons from the previous layer and a layer with  $1/3$  of the neurons from the previous layer.

As observed in fig. 4.5, the addition of extra layer resulted in performance improvements. The 1/3-sized added layer performed almost equally well compared to the 2/3-sized layer. To maintain a simpler architecture, we will opt for the 1/3-sized layer.



**Fig. 4.5:** Comparison of single-layer and two-layer neural networks.

Since the second hidden layer consists of only 4 neurons and the model performs well in this configuration, there is no need to add more layers. Therefore, we will conclude the experiments on this variant.

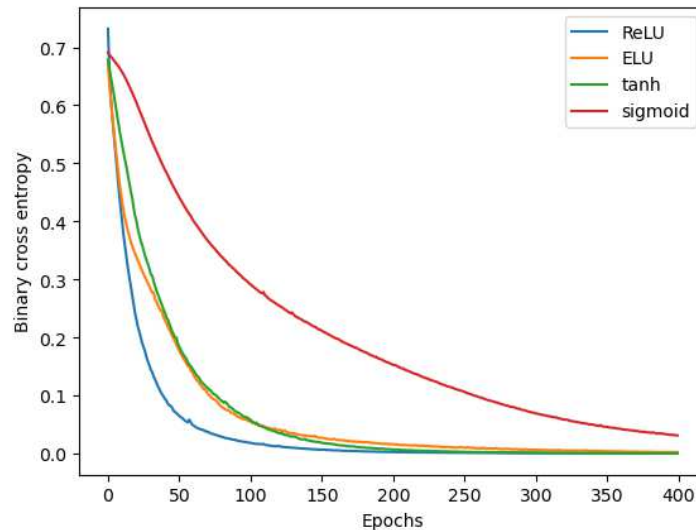
### Model compilation

This step in neural network training involves configuring the secondary model parameters for the training process. It typically includes such components as loss function, activation function, optimizer, regularization and dropout.

Considering that binary cross-entropy is widely recognized as the standard loss function for binary classification tasks in machine learning, and after evaluating other available loss functions in Keras, it was determined that none of them can yield satisfactory results for the current task. Therefore, the decision has been made to stick with binary cross-entropy as the preferred option.

Next, it is time to select the activation functions for the neural network. There are two types of activation functions to consider: those between the input and inner layers, and the one preceding the output layer. For the activation function between the hidden layers, ReLU is a widely adopted choice. It effectively introduces non-linearity to the network by setting negative values to zero while leaving positive values unchanged. In the current neural network, ReLU is being utilized as the activation function during initial tests. Additionally, several modified versions of ReLU, such as Leaky ReLU, can be employed as alternatives between hidden layers. Other options include activation functions like ELU, tanh, or sigmoid.

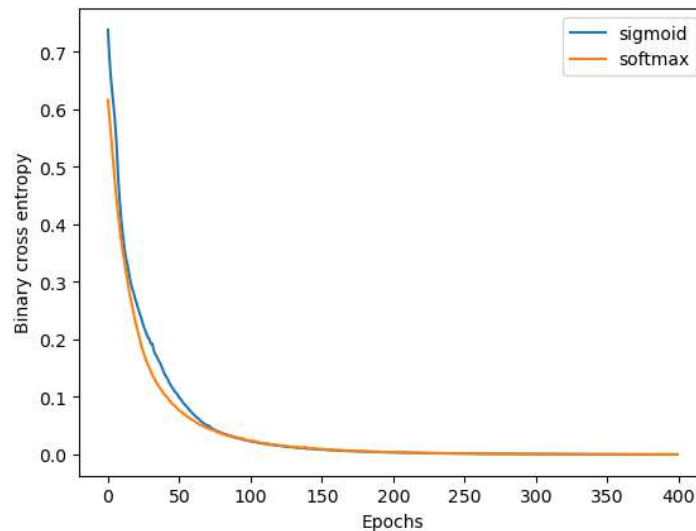
The performance of different activation functions with binary cross-entropy as the loss function is illustrated in fig. 4.6. It is evident that both ELU and tanh achieve a similar level of performance compared to ReLU, albeit with slightly slower convergence. On the other hand, sigmoid demonstrates notably inferior performance compared to the other activation functions. In order to maintain a more conventional approach without unnecessary complexity, we will adhere to the widely adopted ReLU activation function.



**Fig. 4.6:** Comparison of different activation functions of inner layers.

As for the output activation function in binary classification, there are several commonly used options: sigmoid, binary step activation, and softmax. The sigmoid function is widely adopted as a convenient choice for binary classification tasks, as it outputs probabilities between 0 and 1. The binary step activation function, on the other hand, produces a binary output of 0 or 1 based on a threshold. However, it is not differentiable, which limits its direct use in gradient-based optimization algorithms like backpropagation, and it is not available in Keras. Softmax activation, typically used for multi-class classification, assigns probabilities to each class and ensures their sum adds up to 1. Fig. 4.7 illustrates the binary cross-entropy loss for both sigmoid and softmax activations applied before the output neuron. Both sigmoid and softmax activations demonstrate similar performance, but to adhere to the commonly adopted approach, the sigmoid activation function will be utilized.

Next, the selection of a regularization method will be addressed. Regularization techniques are vital in machine learning to mitigate overfitting and enhance the generalization capabilities of models. Overfitting can arise from various factors, including excessive model complexity, training for an extended number of epochs, or limited training data. In this scenario, model complexity is not a concern since the model is relatively simple. To combat the risk of overfitting from excessive epochs, manual control of the number of epochs or early stopping techniques can be employed. However, the primary challenge lies in the limited amount of training data, necessitating the use of a regularization method



**Fig. 4.7:** Comparison of different activation functions of output layer.

that enhances the model's generalization performance.

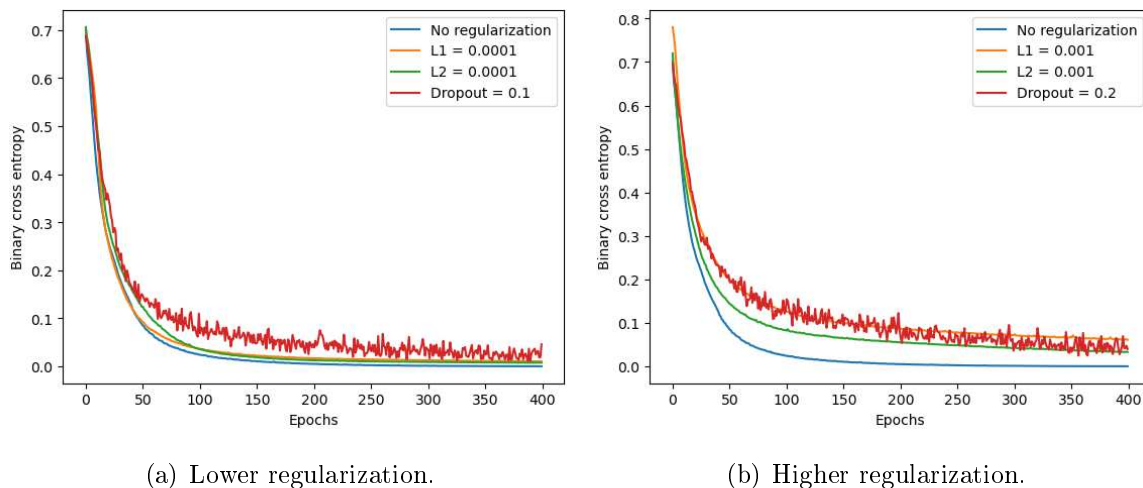
The most commonly used regularization techniques, apart from early stopping, include L1 and L2 regularization, as well as dropout. L1 and L2 regularization are widely used techniques that add a penalty term to the loss function based on the magnitude of the model's weights. L1 regularization encourages sparsity by promoting some weights to become exactly zero, while L2 regularization encourages small weights without enforcing sparsity. These techniques help prevent overfitting by reducing the impact of irrelevant or noisy features. Dropout is a technique where randomly selected neurons are ignored during training. It helps to prevent over-reliance on specific neurons and encourages the network to learn more robust and generalizable representations. Dropout effectively reduces overfitting by introducing noise and promoting ensemble-like behavior among the neurons.

After conducting several experiments, it was observed that even a slight regularization had a negative impact on the model's performance. This can be attributed to the simplicity of the model architecture, which seems to not require an increase in generalization ability. Excessive regularization significantly restricts the model's flexibility and impedes its capacity to effectively learn from the data.

In fig. 4.8, the binary cross-entropy of models with L1 and L2 regularization, as well as dropout, is compared to the model without regularization. The left figure demonstrates the results with lower regularization values, while the right figure illustrates the outcomes with higher regularization values, where the issue becomes more pronounced.

Based on these results, it was determined that the inclusion of regularization did not provide any significant improvements to the model's performance. Therefore, the decision was made to exclude regularization altogether.

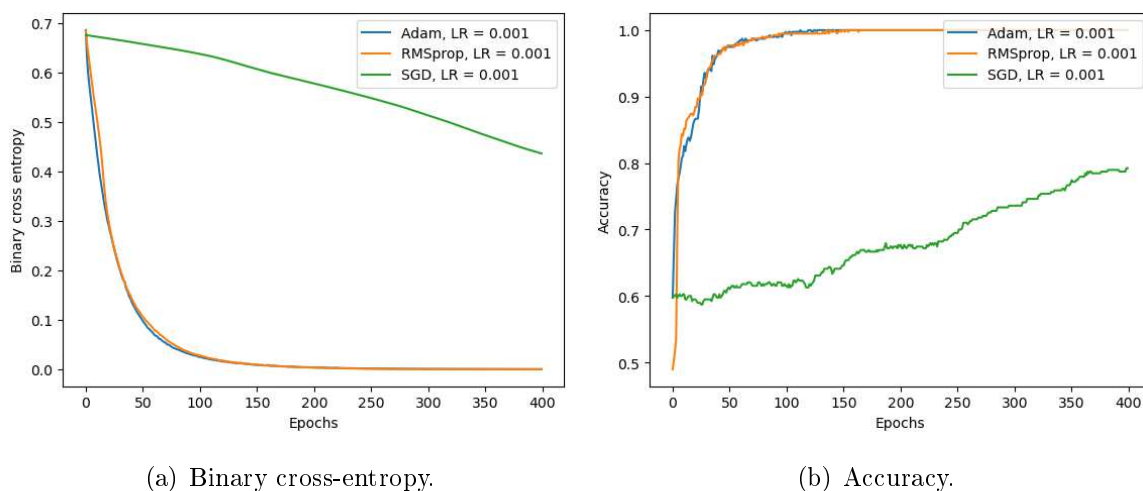
The final step of model setup is to choose an optimizer. An optimizer is an algorithm used in training a neural network to optimize the model's performance. During the training



**Fig. 4.8:** Comparing neural networks with varying regularization types and coefficients.

process, the optimizer adjusts the weights and biases of the neural network based on the computed gradients of the loss function. The goal of the optimizer is to minimize the loss function and improve the accuracy of the model predictions. For all the tests conducted so far, the Adam optimizer was utilized. However, to further explore the performance of different optimizers, other popular options such as RMSProp and Stochastic Gradient Descent (SGD) will be tested and compared.

In Figure 4.9, the loss function during training is depicted for three different optimizers: Adam, RMSProp, and Stochastic Gradient Descent (SGD). The models were trained using a fixed learning rate of 0.001. The plot provides a visual comparison of how the loss function evolves over the course of training for each optimizer.



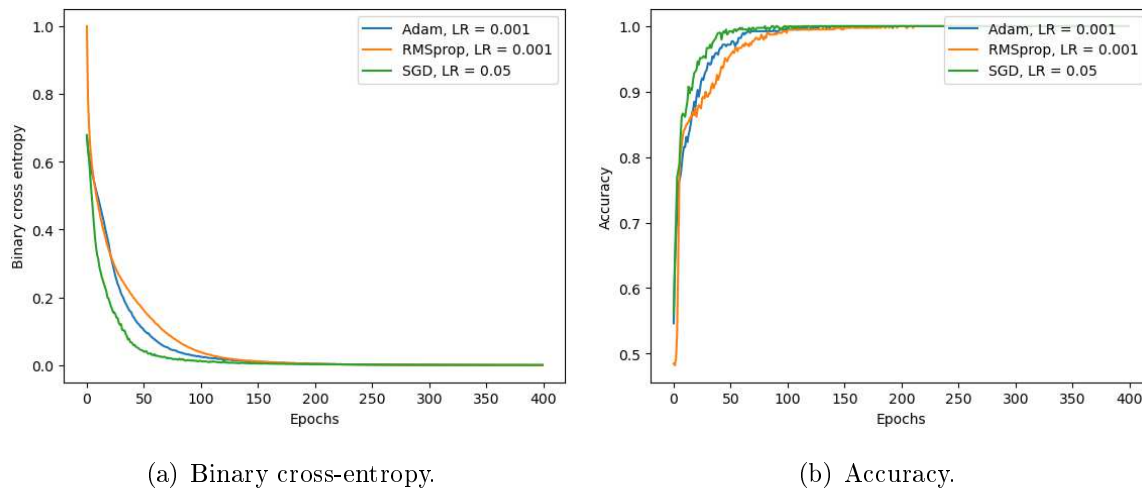
**Fig. 4.9:** Comparing different optimizers with Learning Rate 0.001.

It is important to note that comparing optimizers directly based on a fixed learning rate may not provide a fair assessment, as different optimizers respond differently to learning rate variations. To ensure a fair comparison, it is recommended to select the optimal

learning rate for each optimizer individually and then compare their performance.

The performance of Stochastic Gradient Descent (SGD) can be significantly improved by selecting an appropriate learning rate, as shown in fig. 4.10. In this case, all three optimizers demonstrate comparable performance. Additionally, while the choice of optimizer can impact training speed, in that case it is not a major concern, as the model is simple and the dataset is relatively small.

Considering that all three optimizers perform similarly, the decision is made to use Adam as the optimizer for its established effectiveness and ease of use.



**Fig. 4.10:** Comparing different optimizers with optimized Learning Rates.

## Model training

Once the model is compiled, the subsequent step involves training the model. During this stage, it is essential to determine appropriate hyperparameters such as the learning rate, batch size, and number of epochs.

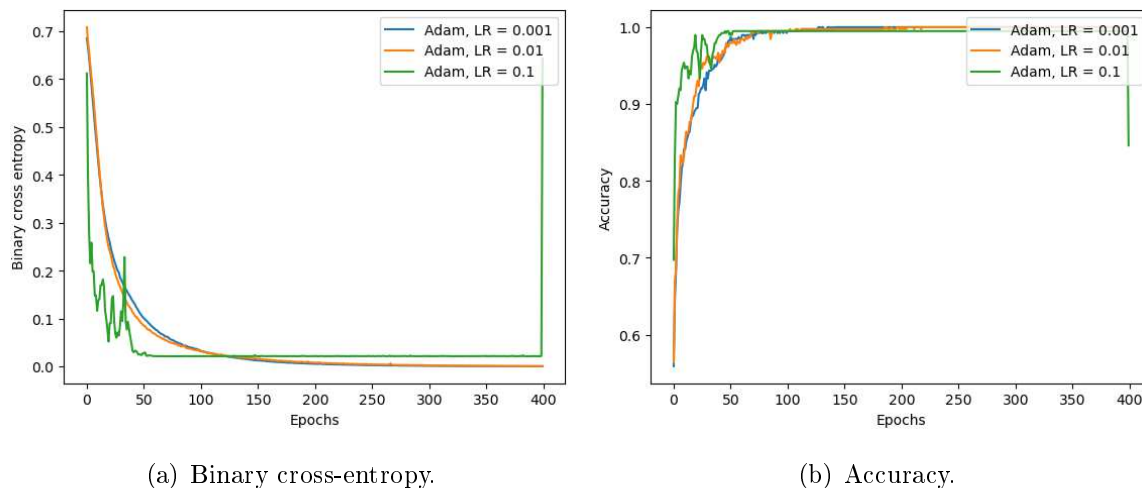
The learning rate determines the step size taken during optimization, influencing the speed and quality of model convergence. It is crucial to choose an optimal learning rate that balances training speed and model accuracy.

In this particular scenario, where the model is simple and the dataset is small, the training speed is not a significant concern. As a result, smaller learning rates can be utilized. However, the learning rate will be optimized for demonstration purposes.

A commonly employed approach involves finding a threshold value for the learning rate, below which the loss function consistently decreases during training, while higher learning rates cause the loss function to either increase or oscillate (fig. 4.11). The aim is to identify the highest learning rate value that ensures a consistent decrease in the loss function from the beginning.

After several iterations of the aforementioned process, it was determined that the maximum learning rate, at which the loss consistently decreases without exhibiting oscillation, is 0.012.





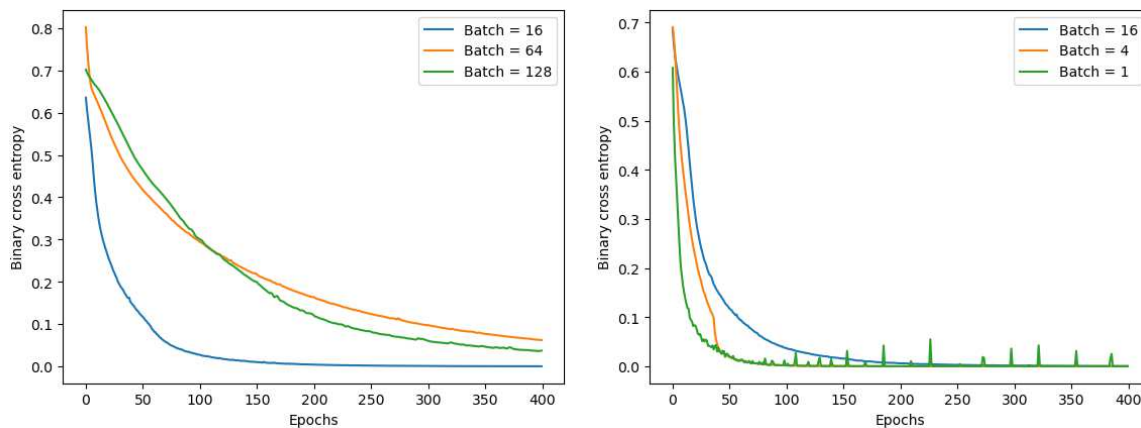
**Fig. 4.11:** Comparing different Learning Rates.

The batch size refers to the number of samples processed in a single iteration during training. A larger batch size can provide computational efficiency but might compromise generalization performance. On the other hand, a smaller batch size may require more iterations but can lead to better generalization. With a small dataset, using smaller batch sizes is generally recommended. Smaller batch sizes introduce more randomness and variability to the optimization process, which can help prevent overfitting and improve generalization. It provides a form of regularization by updating the model's parameters more frequently based on different subsets of the data.

On the other hand, using a batch size that is too small can lead to several issues, including overfitting, slow convergence, and noisy gradients. Overfitting may occur because the model is unable to learn generalizable patterns from the limited information in each small batch. Slow convergence can be a consequence of the reduced amount of data available for each update, requiring more iterations to reach an optimal solution. Additionally, the gradients calculated on small batches may exhibit higher variability and noise, hindering the learning process.

The figure depicted in fig. 4.12 clearly illustrates that when using a larger batch size, the model's loss function converges to a higher value. This observation indicates that the model struggles to effectively generalize the features of the samples, leading to overfitting. In other words, the model becomes excessively tailored to the training data and fails to capture the broader patterns and characteristics present in the data set. In case of too small batch sizes we can observe spikes on the loss function caused by the gradient noise. After conducting several experiments, a batch size of 16 was determined to be the optimal choice.

The number of epochs corresponds to the total number of times the entire training dataset is passed through the model. It affects how many iterations the model undergoes to learn from the data. It is essential to find an adequate number of epochs to ensure that the model has sufficient training iterations without overfitting.



(a) Too large batch sizes.

(b) Too small batch sizes.

**Fig. 4.12:** Comparing different batch sizes.

The number of epochs can be manually determined by assessing the convergence of the loss function or by utilizing early stopping techniques. In this particular case, experimentation revealed that the model reaches its optimal performance without any significant improvements beyond 300 epochs. Therefore, it was determined that 300 epochs would be an appropriate choice for training the model, striking a balance between capturing important patterns in the data and training time.

### Model evaluation

To evaluate a binary classification model, can be used such metrics as accuracy, precision, recall, specificity and F1 Score.

Accuracy measures the overall correctness of the model's predictions by calculating the ratio of correct predictions to the total number of samples.

Precision quantifies the proportion of true positive predictions out of all positive predictions, indicating the model's ability to accurately identify positive instances.

Recall measures the proportion of true positive predictions out of all actual positive instances, indicating the model's ability to capture positive instances.

Specificity calculates the proportion of true negative predictions out of all actual negative instances, indicating the model's ability to correctly identify negative instances.

The F1 score is a metric commonly used in classification tasks to measure the balance between precision and recall. It provides a single numerical value between 0 and 1 that represents the model's overall performance, where a higher value indicates better performance.

These five metrics can be represented in a form of confusion matrix. The matrix displays the counts of true positive (TP), true negative (TN), false positive (FP), and false negative (FN) predictions. The confusion matrix has a square shape, with rows representing the actual class labels and columns representing the predicted class labels. The four quadrants of the matrix are populated with the counts of instances based on the

model's predictions and the true labels.

The confusion matrix of the final version of the model is depicted in fig. 4.13. From the confusion matrix, it is evident that out of a total of 130 samples in the validation dataset, the model accurately predicted 126 samples. There were no false-negative classifications, indicating that the model successfully detected all instances of pointing movements. However, there were four false-positive classifications, which in this case is preferable. It is better for the system to detect pointing movements even when they are not present, rather than missing true pointing movements.

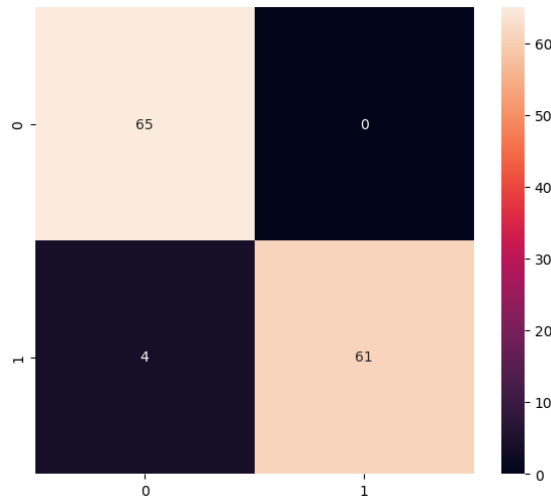


Fig. 4.13: Confusion matrix.

### Model optimization

As almost all of the hyperparameters mentioned in the parts about model architecture design and model compilation are linked, the change of one of them leads to necessity of adjusting others, so all these steps were iterated multiple times to achieve best results.

### Model deployment

Once the deep neural network model for classifying unnatural body positions has been created and trained, it can be integrated into the project. To do so, the model file in .tflite format, which is an extension commonly used for TensorFlow library models, should be copied to the project folder.

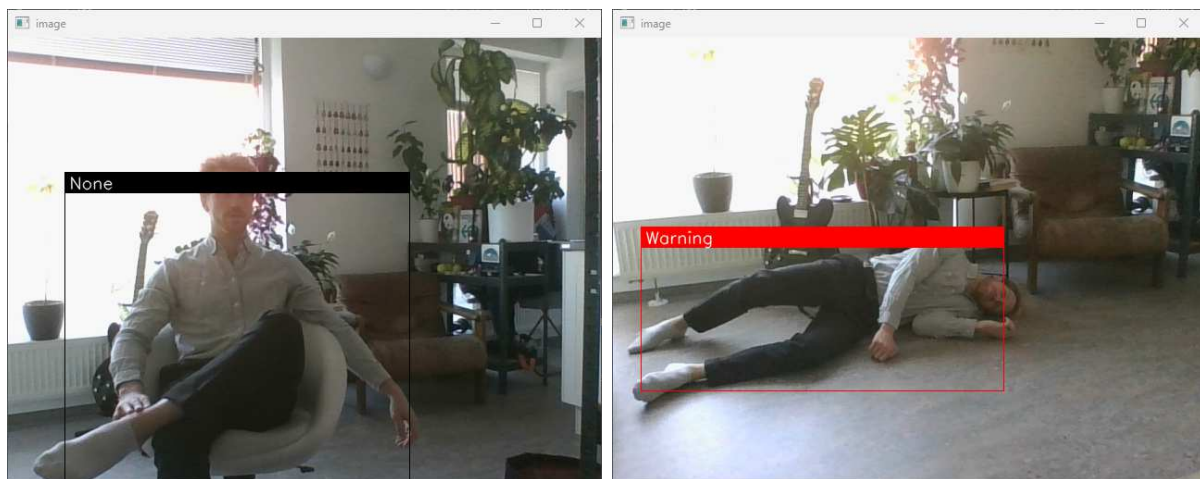
To facilitate the usage of multiple neural networks within the project, a separate module named "keypoint\_classifier.py" has been created. This module handles the initialization, preparation, and data transfer to the neural networks inputs.

Now that all the necessary functionality for working with Tensorflow models has been prepared, the data from the module for determining the position of the human body can be passed to the neural network input. However, before passing the coordinates of the key points to the neural network's input, it is essential to normalize them, as the neural network was trained on normalized data.

Given that all frame-by-frame operations on the input video stream are performed within the `start_cv()` function of the `frame_handler.py` module, it is appropriate to call the `classify_pose()` function within this function to pass the key points to the input of the neural network.

To display the classification results of the user's unnatural body position on the output video stream, functions have been created to add visual elements to the frames. These functions are designed to create a bounding box around the user and add text that displays the classification result. They have been implemented within the relevant module, such as `frame_handler.py`, to ensure seamless integration with the overall video processing pipeline.

The result of the functionality for determining the user's unnatural pose is shown on fig. 4.14.



(a) Normal pose classification.

(b) Unnatural pose classification.

**Fig. 4.14:** Examples of normal and unnatural pose classification.

#### 4.3.2.2 Excluded zones

To address the issue of false alarms caused by the neural network classifying all horizontal positions as unnatural poses, it is essential to differentiate between normal lying positions, such as lying on a bed or sofa, and potentially dangerous or unnatural body positions. This distinction can be achieved by incorporating additional context-awareness into the system.

One of the simplest and most intuitive ways for the user is to designate excluded zones within the frame with bounding boxes. To allow users to define excluded by the bounding boxes and prevent unnecessary actions upon detecting an unnatural body position, a graphical user interface (GUI) from the `bboxes.py` module will be used.

When the `keypoint_classifier.py` module detects an unnatural position of the user's body, the program will verify if the key points of the body are within the designated safe

zone. If so, no action will be taken by the system. However, the user is outside the safe zone, a timer will initiate, and upon its completion, the proper action will be taken.

## 4.4 Smart home device control

As mentioned in the chapter on choosing technologies, the functionality for controlling smart home devices using gestures is based on solving four tasks:

1. Select a device for interaction
2. Determine the position of user's hands
3. Classify a gesture based on hand position data
4. Send a command to a device

### 4.4.1 Device selection

In order to enable interaction with the desired smart home device, a hand pointing gesture will be utilized as the selection mechanism. To implement this functionality, the following information will be required:

1. Position of key points on the user's arms;
2. Position of smart home devices.

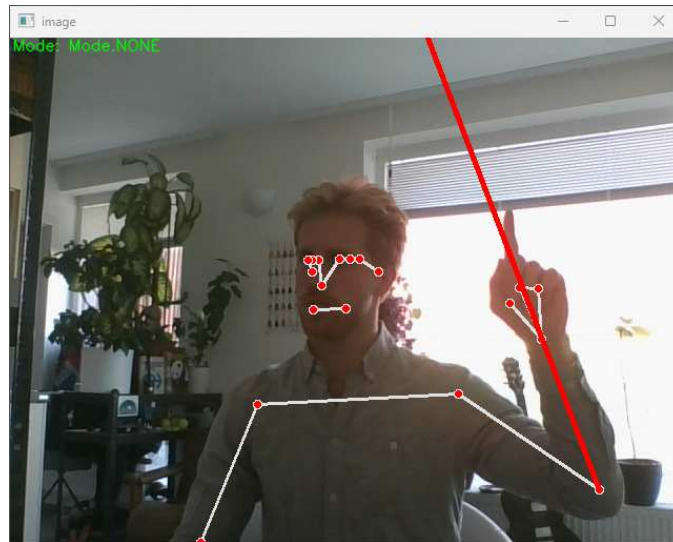
By combining these two sets of information, the system will effectively interpret the user's pointing gesture and facilitate interaction with the desired smart home device.

The human body position module will provide crucial data for determining the pointing direction. Specifically, key points 13, 15, 14, and 16, which correspond to the elbows and wrists of the left and right hands respectively, will be utilized. A beam will be created by drawing a line from the elbow key point through the wrist key point (fig. 4.15). This beam will serve as the indicator for pointing towards smart home devices.

To facilitate geometric calculations of the beams from the key point coordinates, a dedicated module named *geometry.py* will be created. This module will handle the necessary computations, enabling accurate determination of the pointing direction and other geometric operations required for effective interaction with the smart home devices.

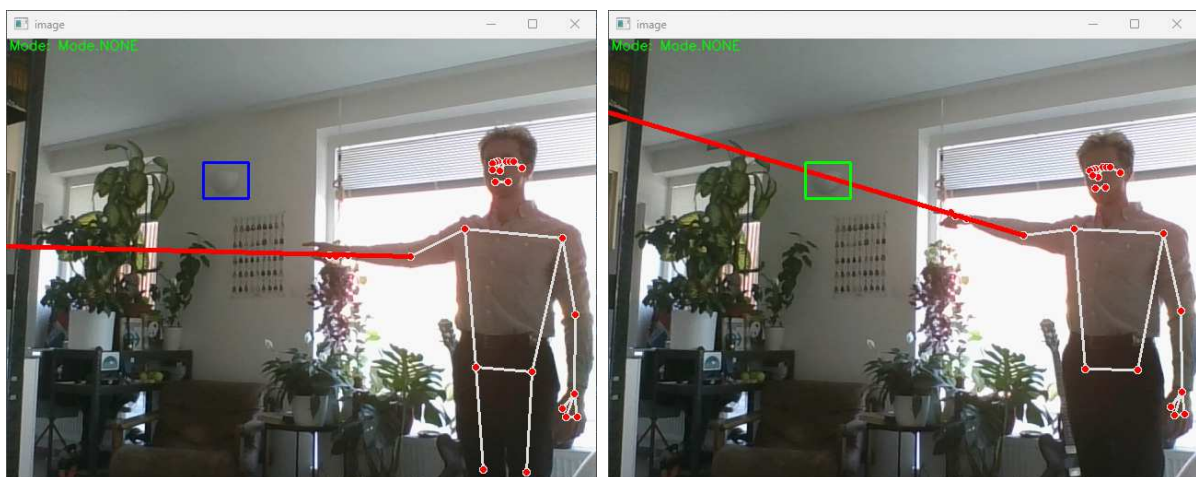
The same functionality employed for designating safe zones and delimiting camera field-of-view into zones will be utilized to identify smart home devices within the frame. The *bboxes.py* module will be utilized to draw bounding boxes around the smart home devices present in the frame.

To determine the interaction target, a new function will be incorporated into the *geometry.py* module. This function will calculate the presence of intersection points between



**Fig. 4.15:** Pointing beam superimposed on the output video.

the bounding boxes and the pointing beam. If any intersection points are detected between a rectangle and the beam, the device located within that rectangle will be considered selected for interaction (fig. 4.16).



(a) Device is not selected.

(b) Device is selected.

**Fig. 4.16:** Device selection by the pointing beam.

During the work on this task, two specific issues have been identified:

1. The First problem pertains to "false positive" device selection. This occurs when the user's hands are in motion during regular activities, causing the pointing beam to inadvertently intersect with device bounding boxes. Consequently, devices may be erroneously selected and triggered, leading to unwanted interactions such as unintentional device activation or deactivation.
2. The Second problem involves inaccuracies in the user's pointing gestures as captured by the camera. While the user may physically direct their hand precisely towards

a device, the camera's perspective may not accurately reflect this alignment. As a result, difficulties can arise when attempting to interact with devices, hindering the user experience.

### **Pointing movement filter**

To mitigate the issue of inadvertent device selection during hand movements, it was decided to introduce a "pointing movement filter" between the module responsible for determining the user's body position and the component that constructs the pointing beam. The primary function of this filter is to differentiate between the user's regular everyday movements and intentional pointing gestures directed at smart home devices. Consequently, the pointing beam will only be generated when the user explicitly intends to interact with a device, effectively preventing accidental selections during routine hand movements. This ensures that the pointing mechanism remains responsive only to the user's intentional actions, enhancing the overall usability of the system.

Several approaches can be explored to implement this filtering mechanism. One relatively straightforward method, as outlined in the Google Pose documentation [17] and discussed in chapter 3, involves calculating parameters such as limb-to-body angles and distances between key points on the user's body. By establishing specific conditions for these parameters, the filter can identify movements indicative of pointing gestures.

However, as was already noted, this approach presents certain challenges. Similar to the issue encountered with recognizing unnatural poses, the variability in these parameters and the complexity of the conditions required to accurately determine pointing movements can create difficulties.

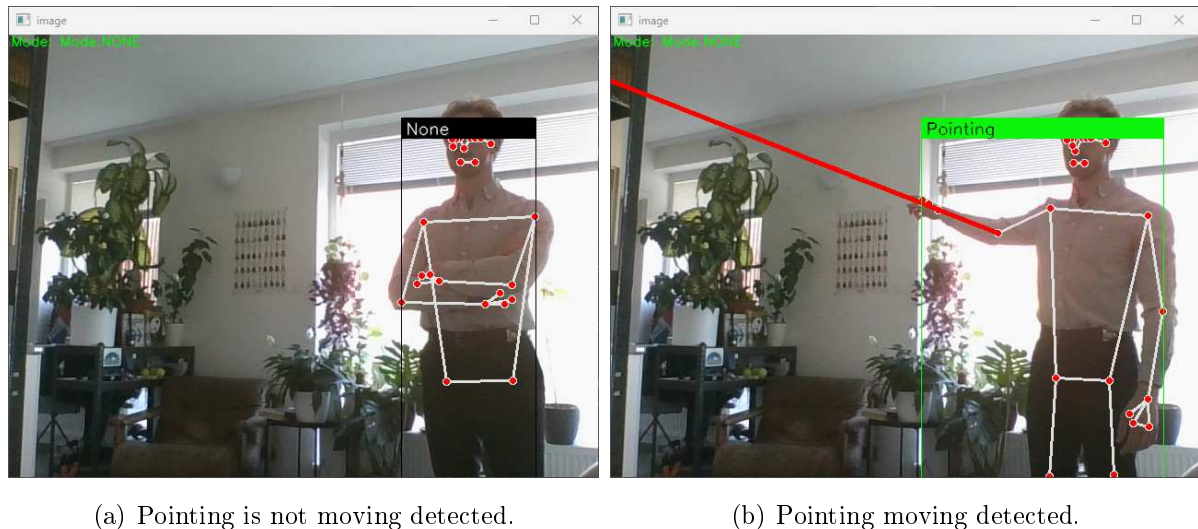
Therefore, while this method may offer technological simplicity, it is essential to carefully address the inherent complexities associated with establishing suitable conditions for accurate pointing gesture recognition.

Given the similarity between the problems of unnatural pose recognition and filtering the pointing movement, a unified solution can be applied to address both concerns. This involves developing a neural network capable of classifying pointing movements by analyzing the relative positions of body key points. The process of creating and training this neural network for the filter will closely resemble the approach described for detecting unnatural body positions, so it will not be described.

By incorporating the filter block into the program, the construction of the pointing beam will now occur exclusively when the filter identifies a genuine pointing movement. This crucial addition effectively mitigates the risk of unintended device selections during hand movements, ensuring a more controlled and accurate interaction process. An illustrative example demonstrating the functioning of the filter is presented in fig. 4.17.

### **Bounding box snapping**

To address the challenge of inaccurate pointing from the camera's perspective, various



**Fig. 4.17:** Example of the functioning of the pointing movement filter.

solutions can be explored. One such approach involves modifying the hardware configuration by adding an additional camera to enable binocular vision or utilizing a dedicated binocular camera. However, implementing this solution would necessitate the development of algorithms capable of calculating the spatial positioning of objects based on binocular video. Additionally, the Google Pose tool may need to be adapted to function with binocular video inputs. Tackling these tasks goes beyond the scope of the current project. Nonetheless, this solution offers a potential avenue for improving the accuracy of pointing gestures by leveraging the benefits of binocular vision technology.

To enhance the user experience with the system, an alternative and simpler algorithm can be implemented for snapping the pointing beam to the device bounding box. This approach aims to improve the accuracy of device selection without the need for complex calculations or additional hardware.

The primary purpose of this approach is to enable device selection and maintain the selected state even in scenarios where the pointing beam does not directly intersect with the device bounding box. This functionality alleviates the need for precise pointing accuracy, simplifying the process of selecting and interacting with devices.

To implement this feature, a new function will be incorporated into the *geometry.py* module. This function will calculate the minimum distance between the pointing beam and the center of the device bounding box. If the calculated distance is below a predefined threshold value, denoted as  $d_{min}$ , the device will be considered selected (fig. 4.18). Conversely, if the distance exceeds another predetermined threshold value, labeled as  $d_{max}$ , the device will be deselected.

After the problem of choosing a device for interaction has been solved, we can tackle the problem of determining the gesture command for the device.



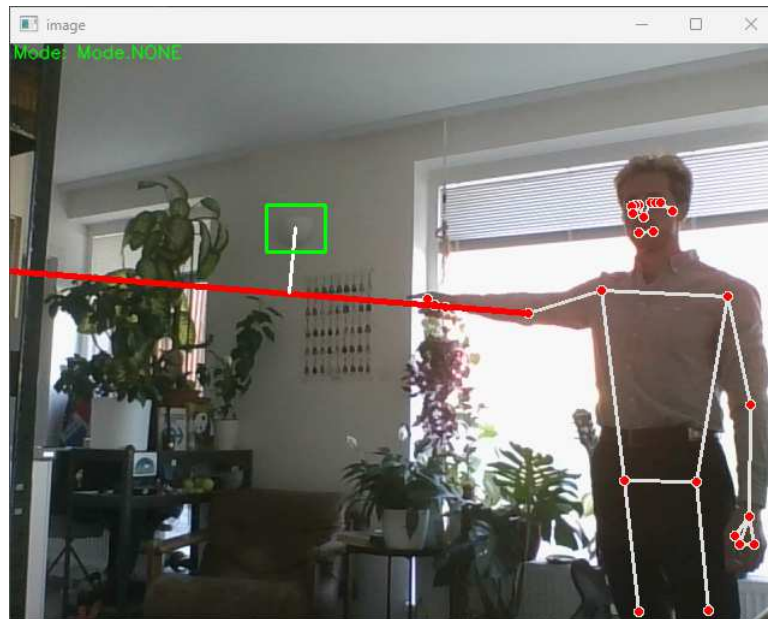


Fig. 4.18: Pointing beam snapped to the device bounding box.

#### 4.4.2 Determining the position of the user's hands

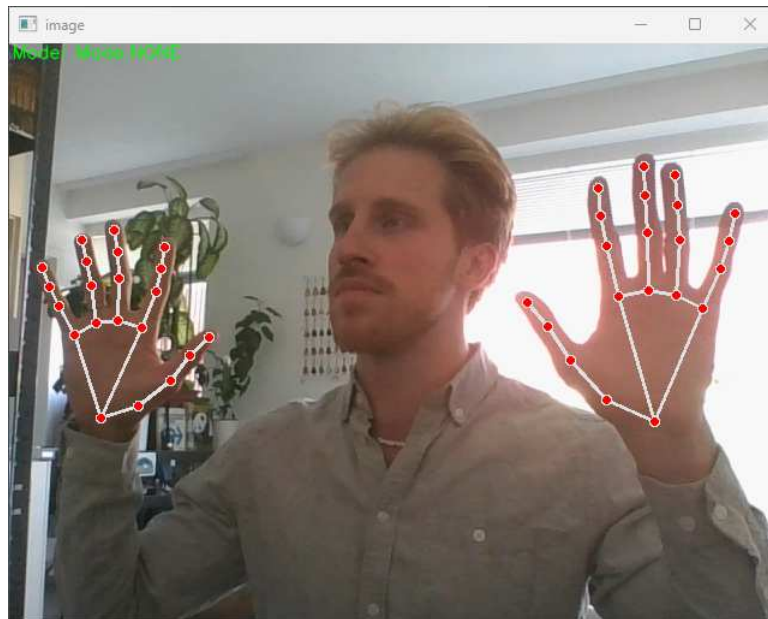
To address the task of determining the position of hands, the Google Hands technology, which is a component of the Google Mediapipe framework, will be employed. Similar to Google Pose, Google Hands utilizes a comparable approach. It takes the input video stream as its input and generates a set of 21 key points for each hand as its output.

By leveraging the capabilities of Google Hands, the system can accurately detect and track the position of hands in real-time. These 21 key points provide valuable information about the hand's pose and spatial configuration, enabling precise analysis and interpretation of hand movements within the system.

The functionality pertaining to working with Google Hands will be encapsulated within a dedicated module called *hands\_processing.py*. This module will handle the initialization and configuration of the Google Hands tool. The process of setting up and configuring Google Hands is almost identical to the corresponding steps for the Google Pose tool. Therefore, these processes will not be elaborated on in detail, as they follow a familiar pattern.

The video stream will be fed into the Google Mediapipe Hands framework using the designated *get\_both\_hands\_landmarks()* method. The resulting output consists of 21 key points for each hand representing the hand joints.

To visualize and present this data, the key points for each hand will be superimposed on the output video as shown on fig. 4.19.



**Fig. 4.19:** Hands key points.

### 4.4.3 Hand gestures classification

After obtaining the data on the hand positions, the next step is to tackle the task of gesture classification. This involves analyzing the spatial configuration of the hand joints and determining the specific gestures performed by the user.

Before proceeding with the implementation of gesture control for a smart home, it is essential to determine a suitable set of simple and well-defined gestures. These gestures will serve as the basis for controlling various devices within the smart home ecosystem.

To enable effective control, a minimum of two distinct gestures are required to facilitate the selection of commands for the devices. Specific actions or commands will be selected based on transitions between these gestures. Additionally, the inclusion of an additional gesture will allow for the implementation of an adjustment function, enabling users to fine-tune settings or parameters within the smart home environment.

Careful consideration should be given to selecting gestures that are intuitive, easily distinguishable, and comfortable for users to perform. This ensures a seamless and user-friendly interaction experience, enhancing the overall usability and efficiency of controlling the smart home devices through gestures.

The open hand gesture involves extending the fingers outward, with the palm facing forward. This gesture signifies a specific action or command within the smart home system, such as turning on or activating a device.

On the other hand, the hand clenched into a fist gesture involves tightly closing the fingers and enclosing the thumb. This gesture represents a contrasting action or command, typically associated with turning off or deactivating a device within the smart home environment.

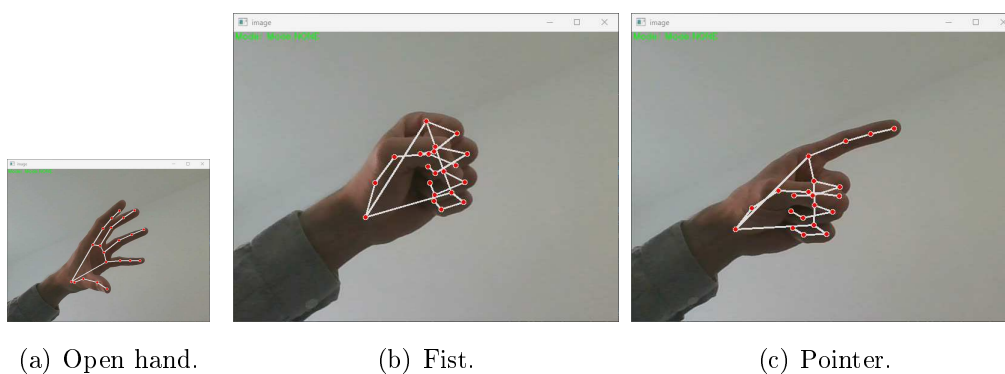
By choosing these contrasting hand positions, the gestures become easily recognizable

and discernible from one another, enhancing the accuracy and effectiveness of gesture-based control for the smart home.

An additional gesture, positioned between the open hand and the hand clenched into a fist, that complements the set is the extended index finger. This gesture involves extending the index finger while keeping the other fingers relaxed or slightly bent.

By including the extended index finger gesture, the set of gestures becomes more comprehensive and enables a wider range of interactions with the smart home system. This allows users to perform precise adjustments and selections, enhancing the overall control and customization capabilities of their smart home devices.

The figure shown below (fig 4.20) illustrates examples of the three gestures along with the superimposed key points representing the hand joints:



**Fig. 4.20:** Examples of gestures for smart home devices control.

Given the similarities in the data format between hand positions and body positions, and the fact that the task of gesture classification differs from classifying unnatural body positions only in the number of key points involved, it is reasonable to implement the solution for gesture classification in a similar manner.

The existing approach for classifying unnatural body positions can be extended and adapted to accommodate gesture classification. The fundamental steps of this solution, such as data preprocessing, feature extraction, and model training, remain largely unchanged. The primary modification lies in the change of the number of neurons on the input and output layers of the neural network.

By leveraging the existing infrastructure and methodology, the solution can effectively handle the classification of gestures by incorporating the hand joint data. This approach ensures a streamlined and consistent workflow, minimizing development efforts and promoting code reuse.

Once the neural network training process, as described in the section about pose classification, has been implemented for this task, the resulting output file in .tflite format is moved to the project folder.

The integration of the gesture classification neural network into the project environment, including its initialization and implementation, has been facilitated through the

functionality provided by the *key\_point\_classifier.py* module.

The results of the gesture classification are shown on the fig. 4.21



(a) Open hand.

(b) Fist.

(c) Pointer.

**Fig. 4.21:** Examples of gesture classification.

Once the task of gesture classification has been successfully accomplished, the next step is to utilize this classification to select commands for a smart home device. This command selection process is achieved through a straightforward algorithm that tracks transitions between gestures.

The algorithm analyzes the sequence of gestures by comparing the previous gesture with the current gesture. By considering the combination of the previous and new gestures, the algorithm can effectively map these transitions to corresponding commands.

The simplicity of this algorithm allows for efficient and real-time command selection, enabling seamless interaction with the smart home devices. It eliminates the need for complex decision-making processes and ensures a user-friendly experience in controlling the devices.

#### 4.4.4 Communication with devices

The final step in the process of controlling smart home devices is sending the appropriate command to the selected device. As mentioned previously, modern smart home devices typically provide a custom API that can be accessed using various programming languages. Utilizing the device's API is the most straightforward and logical method for device management. This API-based approach ensures compatibility and ease of integration with a wide range of smart home devices, regardless of their specific functionalities or manufacturers.

Based on the command determined through the gesture transition, the corresponding command within the device's API will be selected. This selected command will then be transmitted to the device, initiating the desired action or operation.

# 5

## Conclusion

This project aimed to explore the potential of integrating computer vision technology with a virtual home assistant specifically designed for the elderly within the context of The concept of Ambient Assisted Living (AAL). By utilizing computer vision technology within an AAL framework, it becomes possible to address various challenges associated with elderly care, such as monitoring their well-being, ensuring adherence to medication and diet, detecting falls, and identifying early signs of neurodegenerative diseases.

Within the scope of this project, two key functionalities were implemented: fall detection and everyday activity tracking. These functionalities were chosen to address important safety and health concerns faced by older individuals living independently. Additionally, a smart home device control functionality was integrated to simplify user interaction with the smart home ecosystem.

The implementation of all three tasks in this project relies on the utilization of Google Mediapipe, a deep learning-based tool for tracking key points of a person's body in space.

The everyday activity tracking functionality is achieved by continuously monitoring the person's position in space based on the tracked body key points. This allows the system to determine the room or functional zone of the room where the person is located. Additionally, the body key point tracking algorithm serves as a foundation for fall detection and certain aspects of smart home device control. In the context of fall detection, the body key points are used to classify unnatural poses of the person. To accomplish this, a custom deep neural network was developed within the project. Deep learning technology was chosen for pose classification due to the complex and non-obvious patterns associated with normal and unnatural poses, as well as the high variability in the relative positions of body key points. The smart home device control functionality within the project leverages the person's body key points to enable device selection through pointing. Alongside pose tracking technology, hand tracking technology, provided by the Google Mediapipe framework, is employed to facilitate device control through gestures. The classification of hand gestures is accomplished using custom trained neural network. By effectively recognizing and categorizing hand gestures, the system can interpret user intentions and execute corresponding commands for smart home device control.

The final version of the system demonstrates good performance across all three tasks. It exhibits quite precise classification capabilities, accurately identifying instances where a person is lying on the floor, effectively tracks and logs the movements of individuals within a room and allows device control, offering users the ability to interact with specific devices through intuitive hand gestures.

There is still significant room for enhancing the performance of all three solutions. The functionality of everyday activity tracking can be expanded by incorporating an activity classification mechanism in addition to determining the person's location within the premises. This enhancement can be achieved by following the approach outlined in [12] [13] [14]. Another valuable improvement would be the ability to detect unattended electric devices based on the user's activity data and provide alerts or automatically power them off. To enhance the fall detection system, one possible improvement is to increase the quality of classification by gathering a larger training dataset. The smart home device control subsystem can be enhanced in several ways. Firstly, improving the pointing precision can be achieved by incorporating a binocular camera setup or by combining video input from multiple cameras. This approach enables a more accurate determination of the user's arms and device positions in a three-dimensional space, leading to enhanced pointing capabilities. Secondly, expanding the set of gestures associated with commands can greatly enhance the range of control and adjustment options available to the user. By introducing additional gesture-command mappings, the system becomes more versatile and adaptable different kinds of devices.

In addition to the existing functionalities, the system can be further enhanced by incorporating additional features such as fire and smoke detection, as well as unauthorized person detection. By integrating fire, smoke or flood detection capabilities, the system can provide an extra layer of safety and security within the smart home environment. It can monitor for signs of fire, smoke and flood, promptly alert the users, and automatically trigger appropriate actions such as activating alarms, notifying emergency services, or initiating evacuation protocols. Furthermore, the system can be extended to include unauthorized person detection. By leveraging computer vision algorithms and intelligent surveillance techniques, the system can analyze video feeds and identify individuals who are not authorized to be present.

In addition to its applications within the smart home environment, several functionalities of the project can also be adapted and utilized in various other domains. One such area is public spaces design, where the contactless control functionality can play a significant role. By implementing gesture-based control mechanisms, the project can contribute to creating touchless interactions in public spaces such as shopping malls, airports, or public transportation systems. This can enhance hygiene measures, improve user convenience, and reduce the spread of contagious diseases by minimizing physical contact with surfaces. Furthermore, the project's functionalities hold potential in industrial environments, particularly in noisy spaces where traditional communication methods may be

challenging. The fall detection feature can help ensure the safety of workers by promptly detecting and alerting in the event of a fall or accident. Additionally, the contactless control functionality can enable hands-free operation of machinery or equipment, reducing the risk of accidents caused by physical contact.

The complete source code of the project is available at  
[https://github.com/ABeGood/AI\\_assistant](https://github.com/ABeGood/AI_assistant)

# Literatura

- [1] VIMARLUND V, BORYCKI EM, KUSHNIRUK AW, AVENBERG K. Ambient Assisted Living: Identifying New Challenges and Needs for Digital Technologies and Service Innovation. *Yearb Med Inform.* 2021 Aug;30(1):141-149. doi: 10.1055/s-0041-1726492. Epub 2021 Apr 21. PMID: 33882606; PMCID: PMC8416233.
- [2] CICIRELLI G, MARANI R, PETITTI A, MILELLA A, D'Orazio T. Ambient Assisted Living: A Review of Technologies, Methodologies and Future Perspectives for Healthy Aging of Population. *Sensors (Basel).* 2021 PMID: 34069727; PMCID: PMC8160803.
- [3] RITCHIE H, ROSER M . Age Structure. *Ourworldindata.org* [online]. Available from: <https://ourworldindata.org/age-structure>.
- [4] United Nations, Department of Economic and Social Affairs, Population Division (2019). *World Population Ageing 2019: Highlights (ST/ESA/SER.A/430)*.
- [5] GORARD, Stephen, FURLONG, JOHN, MADDEN, LOUISE. (2003). Older adults' use of information and communication technology in everyday life. *Ageing and Society.* 23. 561 - 582. 10.1017/S0144686X03001302.
- [6] SMITH, Aaron. (2014). Older adults and technology use. *Pew Research Centre* [online]. Available from: <https://www.pewresearch.org/internet/2014/04/03/older-adults-and-technology-use/>.
- [7] MURMAN DL. The Impact of Age on Cognition. *Semin Hear.* 2015 Aug;36(3):111-21. doi: 10.1055/s-0035-1555115. PMID: 27516712; PMCID: PMC4906299.
- [8] KIM SY, GIOVANELLO KS. The effects of attention on age-related relational memory deficits: evidence from a novel attentional manipulation. *Psychol Aging.* 2011 Sep;26(3):678-88. doi: 10.1037/a0022326. PMID: 21707178; PMCID: PMC3193860.
- [9] NYBERG L, LOVDEN M, RIKLUND K. Memory aging and brain maintenance, *Trends in Cognitive Sciences*, ISSN 1364-6613.
- [10] YOLO-Pose Multi-person Pose estimation model [online]. Available from: <https://github.com/TexasInstruments/edgeai-yolov5/tree/yolo-pose>



- [11] MediaPipe Hands [online]. Available from: <https://github.com/google/mediapipe/blob/master/do>
- [12] ROSEBROCK, Adrian. Human Activity Recognition with OpenCV and Deep Learning [online]. November, 2019. Available from: <https://pyimagesearch.com/2019/11/25/human-activity-recognition-with-opencv-and-deep-learning/>
- [13] STREELA S R, IDICULA Sumam Mary, Action Recognition in Still Images using Residual Neural Network Features, *Procedia Computer Science*, ISSN 1877-0509.
- [14] SNEHITHA B, SREEYA R. S., MANIKANDAN V. M., "Human Activity Detection from Still Images using Deep Learning Techniques," 2021 International Conference on Control, Automation, Power and Signal Processing (CAPS), Jabalpur, India, 2021, pp. 1-5, doi: 10.1109/CAPS52117.2021.9730709.
- [15] MALLICK, Satya. YOLOv7 Pose vs MediaPipe | Full comparison of real-time Pose Estimation [online]. Available from: <https://www.youtube.com/watch?v=hCJIU0pOl5g>
- [16] COCHARD, David. BlazePose: A 3D Pose Estimation Model [online]. Available from: <https://medium.com/axinc-ai/blazepose-a-3d-pose-estimation-model-d8689d06b7c4>
- [17] Pose classification options: Official Google documentation for developers [online]. Available from: <https://developers.google.com/ml-kit/vision/pose-detection/classifying-poses?hl=en>
- [18] HEATON, Jeff. Introduction to Neural Networks with Java. Heaton Research, Inc. November 25, 2005. ISBN 978-0977320608.