**Т Е Х Н И Ч Е С К И У Н И В Е Р С И Т Е Т – С О Ф И Я**

**Ф а к у л т е т “Приложна математика и информатика”**

**Дипломна работа**

**Тема: „Система за лицево разпознаване и проследяване. Реализация във вградени системи.*“***

***на***

***Симеон Христов Лаплев***

**Факултетен номер: *181217011***

Изработил:............... Научен ръководител.................

/Симеон Лаплев/ / ас. Златко Захариев /

**София, 2021г.**

**Съдържание**

**ГЛАВА 1 - ВЪВЕДЕНИЕ**

* 1. **Обща информация**
     1. **Кратка история**
  2. **Цели и мотивация**

**ГЛАВА 2 - ФУНКЦИОНАЛНИ ИЗИСКВАНИЯ**

1. **Разпознаване на лице и локализация**
2. **Проследяване на лицето със сбор от сензори на вградената система**

**ГЛАВА 3 -** **ТЕХНОЛОГИИ ЗА ИЗГРАЖДАНЕ НА СИСТЕМАТА**

1. **Raspberry Pi 4**
2. **Python 3**
3. **OpenCV**

**ГЛАВА 4 - АРХИТЕКТУРА НА СИСТЕМАТА**

1. **Механична структура и базов набор от сензори**
2. **Софтуер**

**ГЛАВА 5 – ПРИНЦИП НА РАБОТА**

1. **Каскаден класификатор базиран на Хаар черти**
2. **Хистограми на Локални Двоични Черти – Алгоритъм за сравнение**

**ГЛАВА 6 – ДЕМОНСТРАЦИЯ**

**ГЛАВА 7 – ЗАКЛЮЧЕНИЕ**

**ДОПЪЛНИТЕЛНИ РЕСУРСИ И ПОЯСНЕНИЯ**

**ИЗПОЛЗВАНА ЛИТЕРАТУРА И ИЗТОЧНИЦИ**

Глава 1 – Въведение

# Обща информация

По дефиниция **система за лицево разпознаване** е технология пригодена за откриване на човешко лице от дигитално изображение (в частен случай кадър от видео). В зависимост от изискванията на системата, откритото лице може да се сравни с набор от записани лица в база данни и да се идентифицира – например с цел автентикация. В последно време системите за лицево разпознаване са станали изключително широко разпространени. Виждат огромно приложение в днешните смартфони и други типове технологии тясно свързани с роботиката. Тъй като компютърното лицево разпознаване включва измерването на човешки физиологични характеристики, то се категоризира като „биометрика“. Въпреки грубата прецизност на технологията в сравнение с други биометрични системи, лицевото разпознаване се прилага екстензивно в днешно време поради удобството на безконтактността, която е значително предимство на метода. В резултат, лицевото разпознаване се използва в области като: *сложни интеракции човек-компютър*, *видео наблюдение*, *автоматизирано индексиране и категоризация на изображения*; Използва се екстензивно и от органи на реда.

# Кратка история

Автоматизираното лицево разпознаване започва началото си през 60-те години на 20-ти век. Имена като Уди Бледсоу, Хелен Чан Волф и Чарлз Бисон работят по задачата на лицево разпознаване с компютри. Първите им опити за разработка на технологията е била значително опростено сравнение на дистанции между физиологични черти с помощта на компютър. Реалната идентификация на основните черти и техните позиции се е случвало с човешка помощ. Чертите на лицето са се посочвали от доброволци и координатите на съответните позиции са се записвали в база данни. След това компютъра минава през посочените черти в дадено изображение и сравнява дистанциите срещу записаните в базата данни. Това е ставало сравнително бавно, защото е зависело от механичен човешки труд.

След 90-те години на 20-ти век технологията започва да се проучва по-интензивно с подкрепата на американски военно-центрирани академични организации като DARPA и ARL. Интересът е завишен под консенсуса да се разработи „способност за автоматично лицево разпознаване“ за продуктивни приложения в истинския свят под претенцията: „подпомагане на охраняемостта, разузнаването, и органите на реда в изпълнение на техните длъжности.“ Не отнема дълго време разработените системи да видят използваемост, подпомагайки с индексиране на снимки върху лични документи с цел откриване на злоупотреби. До 1990-те системите за лицево разпознаване се разработват основно използвайки фото портрети на човешки лица. Проучването върху проблема с откриване на лица надеждно в образ, който съдържа и други обекти, се засилва през ранните години на същото десетилетие с така наречения *принципен компонентен анализ*. Метода се разработва от Матю Търк и Алекс Пентланд. Това решение приема идеята, че всяко едно уникално откриваемо лице може да се представи като тегловен процент на чертите от лицата записани в тренировъчния базис. Системата се оказва доста ефикасна със сравнително малък брой тренировъчни данни. Тренировъчните данни, въпреки това, изискват сравнително стриктна нормализация и осветление, което по това време не е било много изгодно за изчисление. Метода има и недостатъка, че е лабилен към не добре осветени примери. Промяната в емоционалното изражение на лицето също представлявало спънка за алгоритъма.

През 2001-ва година откриването на лица в истинско време посредством видео става възможно благодарение на „Виола-Джоунс методологията за откриване на обекти.“ Паул Виола и Майкъл Джоунс комбинират техния метод за откриване на лица с **подхода на Хаар наподобяващи черти** за откриване на обекти в дигитални изображения. Явява се така нареченият *AdaBoost*, първият фронтален лицев детектор, работещ в истинско време. До 2015-та година алгоритъма на Виола-Джоунс се имплементира във все по-малки и по-маломощни ръчни устройства и вградени системи. Съответно, Виола-Джоунс алгоритъма не само разширява практическото приложение на системите за лицево разпознаване, но и ги полага в центъра на вниманието, що се отнася до приложимост в преобладаващите технологии на модерния свят.

# Цели и мотивация

В днешно време достъпността на дребен, но същевременно способен хардуер е установена като факт. С един микроконтролер, човек има възможността да интегрира сложни концепции в система с най-различни приспособления. Целта на тази работа е да установи една базова интеграция на сравнително сложна концепция – именно лицево разпознаване и локализация. Да покаже, че приложността и разширяемостта на един подобен проект няма лесно начертана граница. Благодарение на технологиите, които предстои да се разгледат, ще се демонстрира достъпността на подобни разработки.

Когато се замислим каква е приложността на подобна интеграция, много примери излизат наяве. С оглед на простота на текста, ще дадем два конкретни примера.

*Пример 1.* Да кажем, че имаме определена личност под наблюдение с медицински цели. Въпросната личност се намира в помещение, в което е свободна да обикаля. Ключова информация се състои в образа и изражението на лицето на човека. Ключови кадри трябва да се събират с възможно най-голяма честота. Тази информация трябва да се снема по начин, по който ще е нужна възможно най-малко нормализация на данните. Проблемът е, че помещението е сравнително голямо и за това за възможно най-качествено изображение в далечина, фокусното разстояние на използваната леща е голямо. По този начин полезрението на камерата и/или другите сензори в конструкцията е доста ограничено. Не може лесно да се обхваща цялото помещение и да се очаква чист и постоянен образ на лицето на човека. Тази конструкция е важно да няма слепи точки и за това трябва да има поне две степени на свобода при насочване и да се намира на установена отворена позиция в близост до средата на помещението. Ако субекта е с лице към сензорите и се намира в полезрението на обектива, образа активно ще се центрира около лицето на въпросната личност. Ако лицето не е в полезрението на обектива, сборът от сензори ще почнат да преглеждат панорамно цялото помещение, докато лицето не бъде открито. По този начин се осигурява нужната ключова информация с възможно най-голяма честота в реално време.

*Пример 2*. Имаме нужда от система за домофон с видео връзка, която да има редица интеракции с потребителски разработен интерфейс състоящ се от много функционалности. За целта обаче ни трябва да се разработи сглобка, която да има свободно движеща се камера, която да центрира образа около всички лица в полезрението на домофона. Подобно на системата от *Пример 1* трябва да имаме механично проследяване и опцията за преглед на околностите със сглобката на въпросната камера. В случая, на разработчика му е нужно да състави интеграцията с налични технологии, за да има възможно най-голям контрол върху притока и нормализацията на данни от системата.

Мотивацията за проект с подобна интеграция може да варира драстично. От спонтанна идея за потребителски зададен характер на система за сигурност до нормализация на лицеви данни с академични или медицински цели. Границите на приложимостта се определят от въображението и/или работните изисквания на разработчика.

Глава 2 – Функционални изисквания

# Разпознаване на лице и локализация

Една от основните задачи на система за лицево разпознаване е да разграничи лицето в предоставения дигитален образ. В нашия случай изображението ще представлява кадър от видео, който се обработва в реално време. Трябва лицето да може да се разграничава при умерени светлинни условия с лека вариация в наклона и усукване на образа на лицето. С установена важност е това да се случва поне два пъти в секунда след всички допълнителни обработки върху кадъра.

В детайл, информацията която ни трябва от разграниченото лице е под формата на три различни стойности:

* **Двумерен вектор**  описващ координатите на пиксела върху изображението, от който може да се опише правоъгълния регион от образа, съдържащ лицето.
* Широчината на региона с лицето **w**
* Височината на региона с лицето **h**

В частност, като допълнение към задачата имаме възможността да предвидим притежателя на лицето ако имаме съществено трениран модел и предположението оправдава определена граница на сигурност. В такъв случай ще получим и **име на притежател** като текстови данни.

С получаването на горепосочената информация, имаме всичко нужно да правим допълнителна обработка на сегашния кадър от видеото, както и да манипулираме механизмите на вградената система в зависимост от интерпретацията на тези данни.

Искаме това да се случва с възможно най-малка вариация във времената за обработка, но това основно зависи от бързината/производителността на компютъра, който ще се използва за проекта. Вземайки на предвид някои прости оптимизации, избраният ни компютър ще оправдае няколкократно нужната производителност за тази задача.

# Проследяване на лицето със сбор от сензори на вградената система

След ключовата задача за разпознаване на лицето и неговата локализация, следва не по-малко важната задача за проследяване и центриране на лицата в образа ни.

Тук ключовите изисквания са свързани основно със задвижването на сензорите прикачени към системата. В нашия случай имаме единия базов сензор – камерата. Моментално става ясно, механизма носещ сензорите трябва да има способността да се насочва ъглово с поне две степени на свобода.

Обективът на камерата трябва да се намира в централната точка на въртене, за да няма перспективно изкривяване на образа при завъртане около двете си оси. Иначе казано, разстоянията между обектива и двете крепежни точки по съответните оси на ротация трябва да са равни.

Допълнително, ротацията на обектива трябва да се случва възможно най-плавно, за да не се прескачат ключови кадри и да не се представя ненужно лашкане.

Възниква и въпроса какво се случва при липса на открито лице във видео потока. Много подобно на споменатия *Пример 1* в първа глава, системата трябва да мине в режим на „сканиране“, при който камерата започва да се движи по предварително зададен ъглов маршрут с цел да се открие лице. В момента в който лице бъде открито, системата трябва пак да мине в режим „проследяване“ и да се стреми да центрира лицето подобаващо в кадър.

Конструкцията и крепежните точки трябва да са достатъчно здрави, за да елиминират излишни вибрации от ротацията по двете оси. Тъй като масата на товара, който се носи не е голяма (от порядъка на няколко грама за този проект), то има доста валидни варианти за материалите и крепежните елементи. Подхода, който е поет за тази конструкция ще се обсъди подробно в **Глава 4**.

Глава 3 – ТЕХНОЛОГИИ ЗА ИЗГРАЖДАНЕ НА СИСТЕМАТА

# Raspberry Pi 4

За сърцето на изчисленията бе избран един от най-обещаващите съвременни едноплаткови компютри с възможност за дигитално управление – Raspberry Pi 4**(2)**. Този ARM**(3)** базиран микроконтролер е повече от идеален за целите на тази система. Операционната система, която ще използваме е сравнително олекотена версия на Linux**(4)**. Съответната Linux дистрибуция е базирана на Debian и предоставя нужните зависимости за ефективното използване на възможностите на платката.

*Фиг. 1 – Raspberry Pi 4*

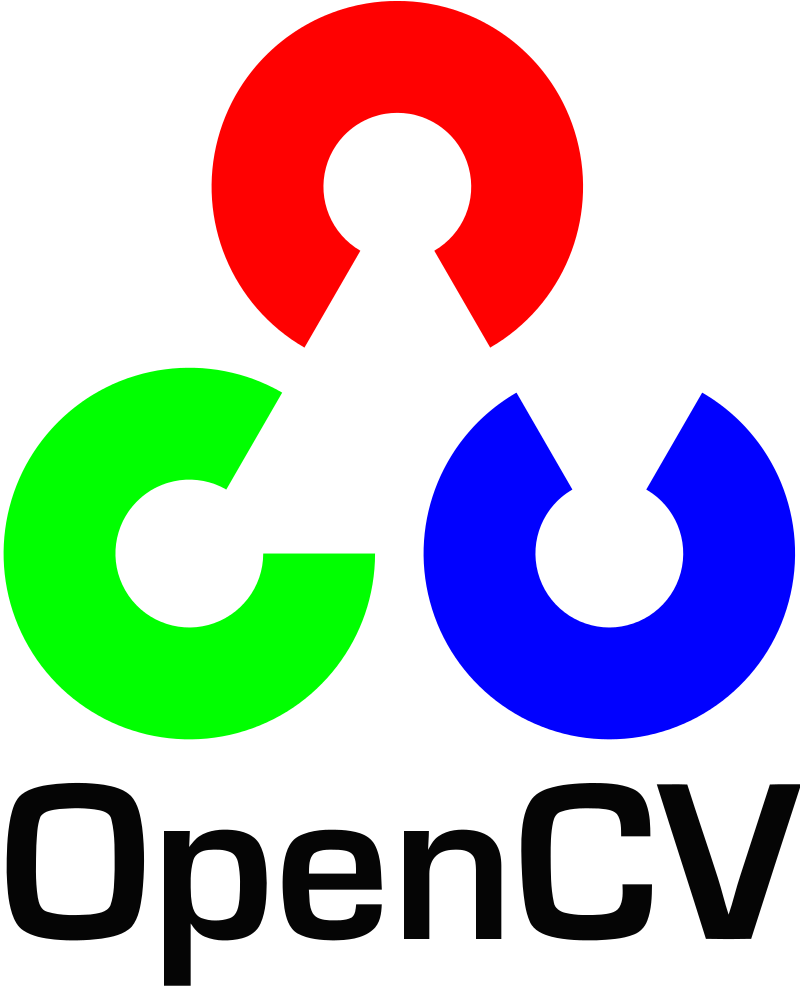
# Python 3

Raspberry Pi фондацията**(5)** предоставя инструменти за ползване на основния програмен език за платформата – Python**(6)**. На пръв поглед този избор е спорен, тъй като основната цел на този компютър е да се използва във вградени системи. Python е език със сравнително голямо изпълнително време за повечето инструкции на високо ниво, тъй като в много от структурите на езика липсват педантичните компилаторски оптимизации, които можем да видим в някои по-общоприети езици използвани в тази сфера (пр. C, C++, Rust и пр.) По-общо погледнато обаче това съвсем не е лошо решение от гледна точка на достъпността за разработчици от всички инженерни и информатични сфери. Python е изключително достъпен и добре поддържан език с голям набор от библиотеки за всякакви цели. Основната идея на използването на подобен език е, че в повечето случаи, разработката е ускорена значително поради липсата на много инженерни специфики, които да касаят разработчика. В следствие, четимостта на софтуера е подобрена. Липсва нуждата от изграждане на посредствено скеле от код, което в крайна сметка е можело да го няма като цяло ако се крие зад нужните абстракции. Конкретиката не бива да е основно притеснение на разработчика, чието внимание следва да е центрирано изцяло върху приложните цели на системата.

Настрана от това отстъпление, Python е изключително способен и изчистен език, центриран върху производителност, макар и не от най-бързите. Компютърът който ще използваме в тази работа има повече от нужния капацитет да изпълнява код на високо ниво, без това да се отрази значително върху крайния резултат.

Редица Python зависимости**(7)** ще бъдат използвани за програмирането на системата. Тяхното инсталиране ще бъде тривиално благодарение на вградения мениджър на пакети, който идва с Python 3 на име **Pip(8)**.

# OpenCV



Основната библиотека в центъра на цялата система е OpenCV**(9)**.

OpenCV (Open Source Computer Vision Library) е библиотека основно насочена към областта на компютърно зрение. Тя бива разработена от Интел и те остават основен поддръжник на проекта и до днес. От 2011 насам, OpenCV поддържа ускорение с графичен хардуер за по-представителна работа в реално време.

Софтуера е разработен за свободно ползване, лицензиран под общите условия на Apache лиценза. Всеки, който се интересува от сферата на компютърно зрение и/или машинно обучение**(10)** може свободно да използва тази мощна библиотека, да я модифицира и преустановява както пожелае.

OpenCV има много възможности вградени в себе си, но за целта на сегашната задача, ние ще се интересуваме главно от модула за разпознаване на лица.

Благодарение на екстензивната ѝ функционалност, можем сравнително лесно да изградим основата на проекта като прозореца на програмата за демонстрация, заедно с по-голямата част от обработката на образа.

Глава 4 – Архитектура на системата

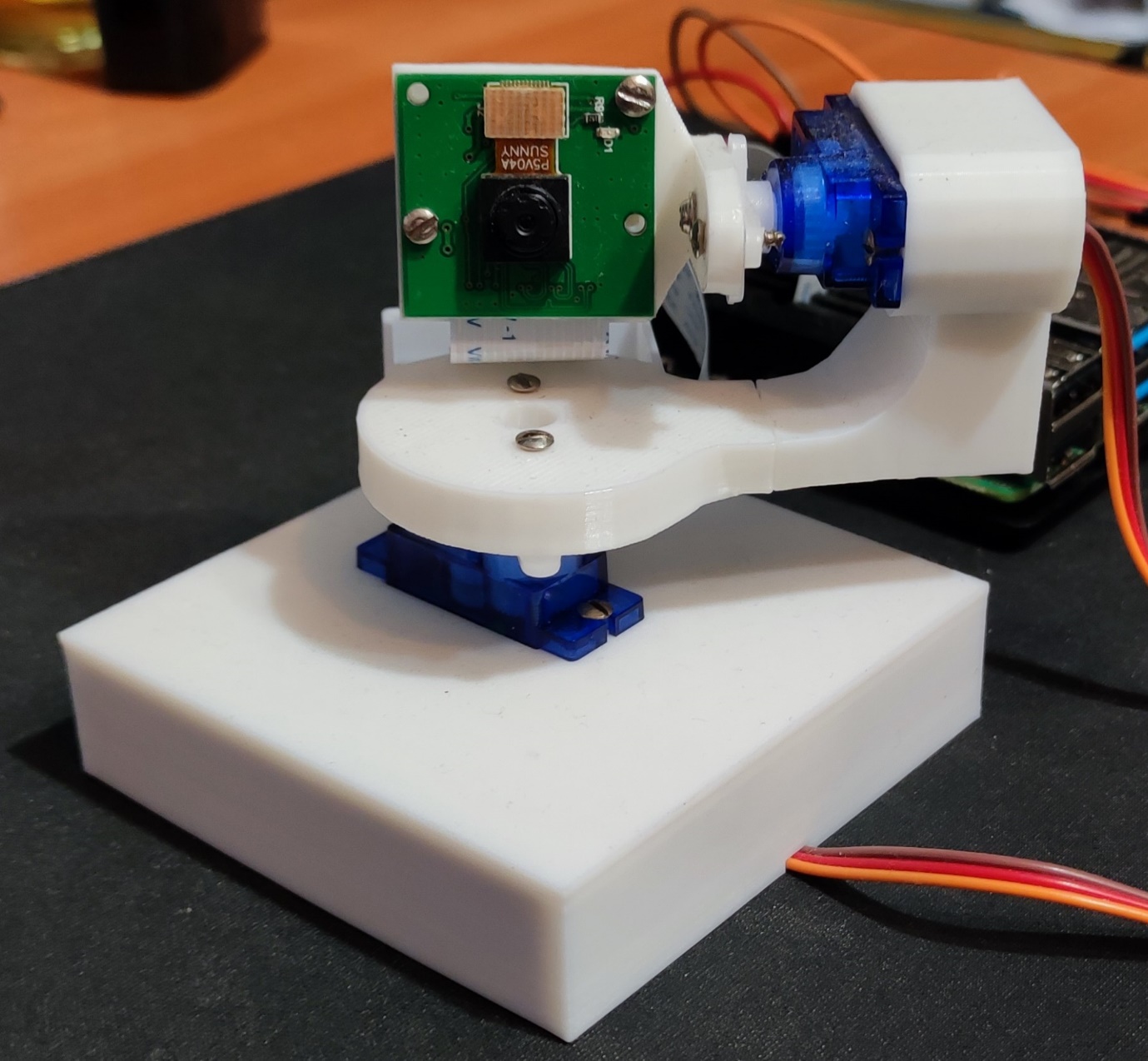
# Механична структура и базов набор от сензори

Конструкцията на механизма трябва да спазва функционалните изисквания подчертани в Раздел 2.2

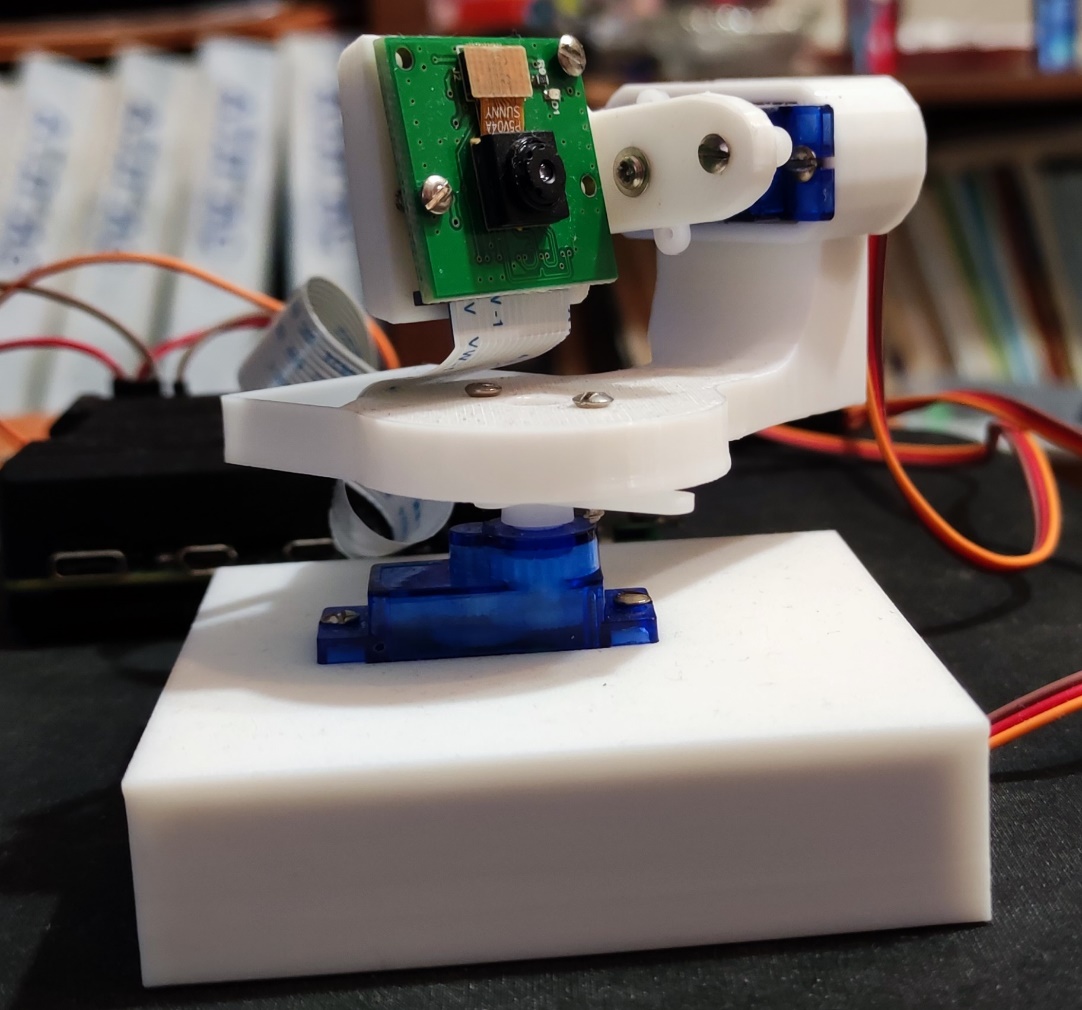
За пълна яснота нека разгледаме подробно готовата конструкция и процеса на произвеждане на механизма.

Механизма е проектиран и създаден от нулата от мен с помощта на CAD**(11)** програмата **Blender(12)** и е подготвен за 3D принтиране**(13)** с **CUDA Slicer(14)**.

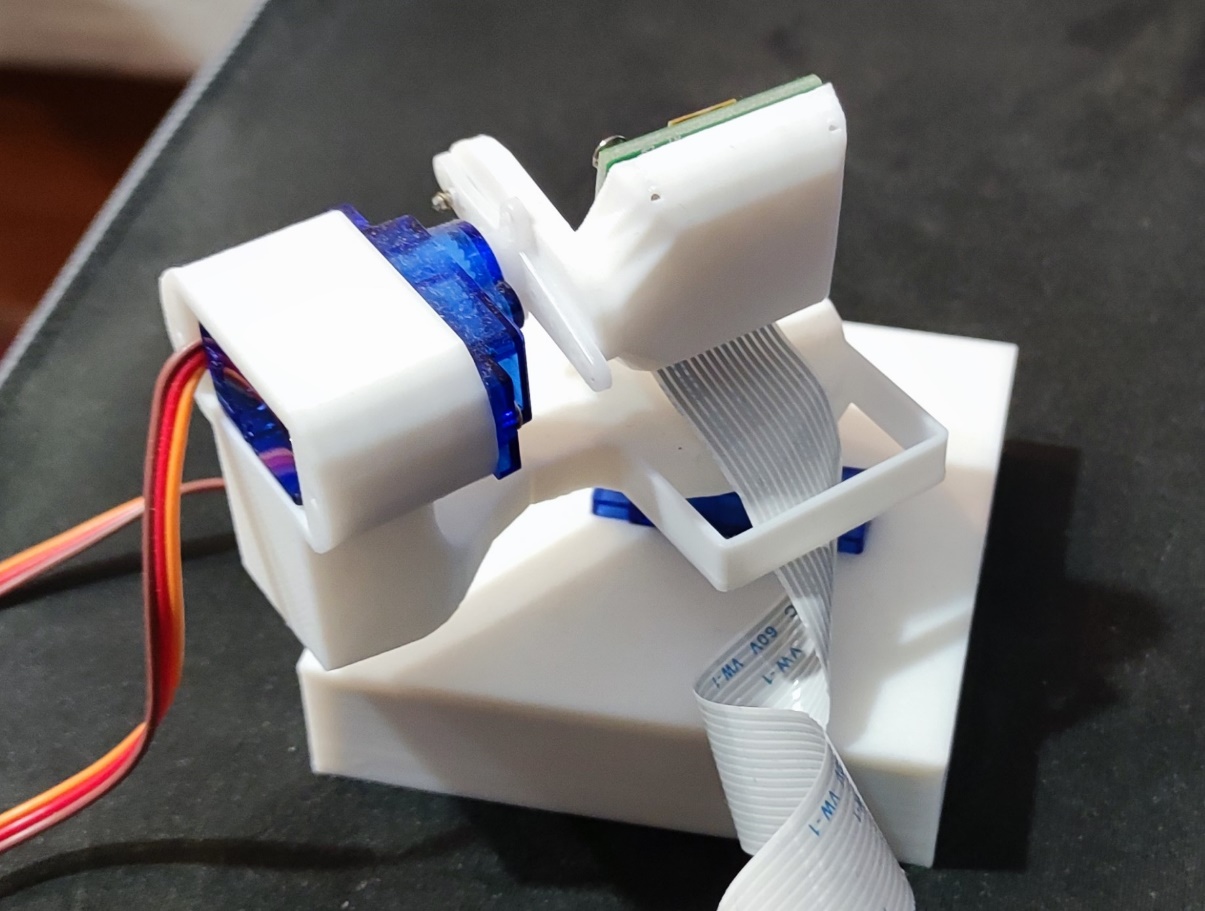
*Фиг. 2.1 – Готовият механизъм (отпред)*



*Фиг. 2.2 – Готовият механизъм (отстрани)*



*Фиг. 2.3 – Готовият механизъм (отзад)*

**

Първият компонент, който ще вземем под внимание са именно двата сервомотори**(15)** *(за улеснение ще се наричат просто „сервота“)*, които ще задвижват конструкцията. Конкретно, нека опишем продукта и да минем през някои негови характеристики. Също така предстои обяснение на приложението му в контекста на тази задача.

*Фиг. 3 – Micro servo SG90*

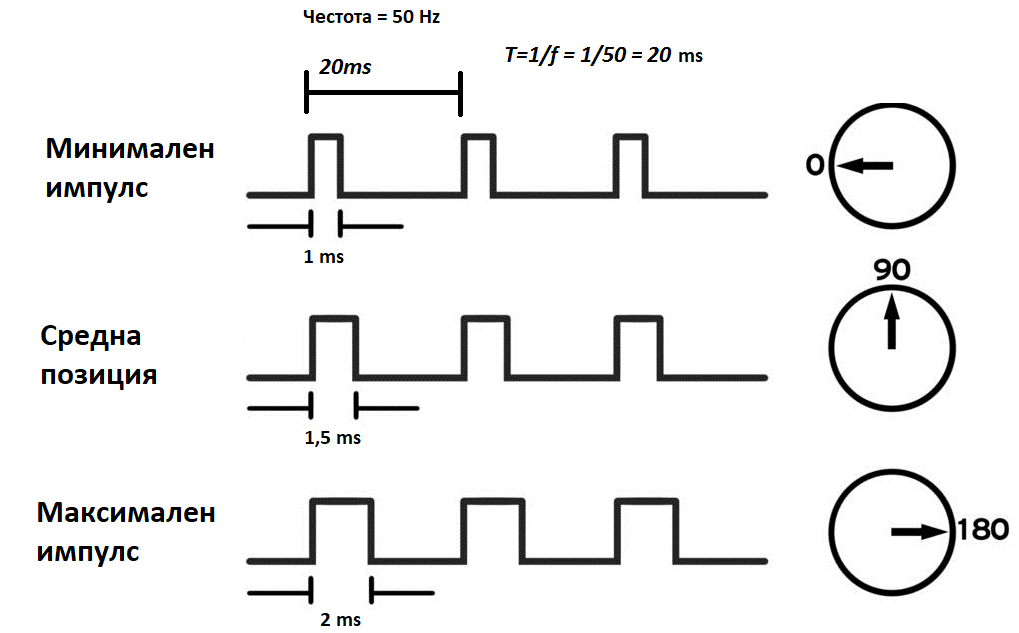


Базовите характеристики на сервото са следните:

* Оперативен волтаж – Между 3 и 6 волта
* Оперативна скорост – От **0,12 сек / 60 градуса** при 4.8 волта захранващо напрежение до **0,1 сек / 60 градуса** при 6.0 волта захранващо напрежение (без прикачен товар)
* Максимален поддържан въртящ момент – Около 1 кг / см
* Ъглов диапазон – 180 градуса
* Импулсна мъртва точка – 7 микросекунди

Сигналът нужен за задаване на ъгъл на ротация е модулиран дигитален сигнал (импулсна модулация)**(16)** с период от 20 милисекунди като ъгъла се определя от широчината на импулсите в този период. Широчината на импулса може да варира в диапазона 1-2 милисекунди. Съответно ако широчината е 1 милисекунда, ъгъла е минималния за диапазона на въртене. При импулс с широчина 2 милисекунди, ъгъла е максималния за диапазона. При 1,5 милисекунди, зададения ъгъл е в средата на диапазона.

*Фиг. 4 – Форма на управляващия сигнал*

**

Ключово е да се отбележи, че ъгловата резолюция на сервото зависи както от импулсната мъртва точка, така и от предавателния механизъм, заедно в всякакви загуби, които той може да ни представи. С други думи, факта че на теория имаме резолюция от порядъка на части от микросекунди когато боравим със сигнала, това не задължително означава, че такава малка промяна в широчината на импулса ще се отрази във физическо движение на моторната ръка. Всяко серво има характерна „импулсна мъртва точка“, това е минималната времева промяна в импулса на която мотора всъщност отговаря с физическо движение. В нашия случай се използва хоби серво с не много добра отзивчивост. Както е посочено в спецификациите, импулсната мъртва точка е 7 микросекунди в нашия случай. Сигналния диапазон с който работим е 1000 микросекунди и имаме общ ъгъл с големина 180 градуса в този диапазон. Вземайки на предвид и минималната отзивчивост от 7 микросекунди, то след проста сметка можем да намерим частта от целия ротационен диапазон, която отговаря на минималния ъгъл на отзивчивост на сервото:

градуса

За нашите цели, тази ъглова резолюция е напълно достатъчна, макар и не идеална ако целим абсолютно плавно въртене.

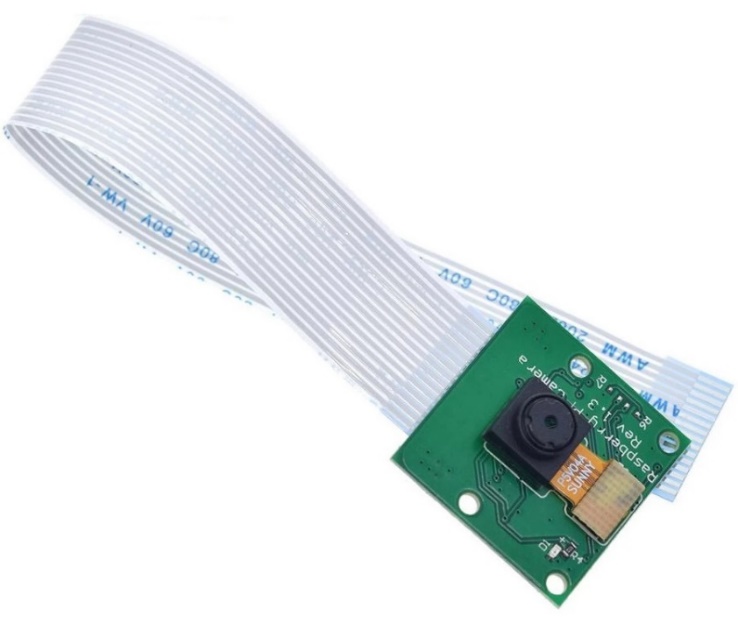
Самият механизъм в цялост се дели на две категории части: **активни** и **пасивни**.

**Активните части** в този случай са камерата прикачена към системата и двете нужни сервота за въртене по двете определени оси.

**Пасивните части** са всички поддържащи и корпусни части, заедно с крепежните елементи за извършване на сглобката.

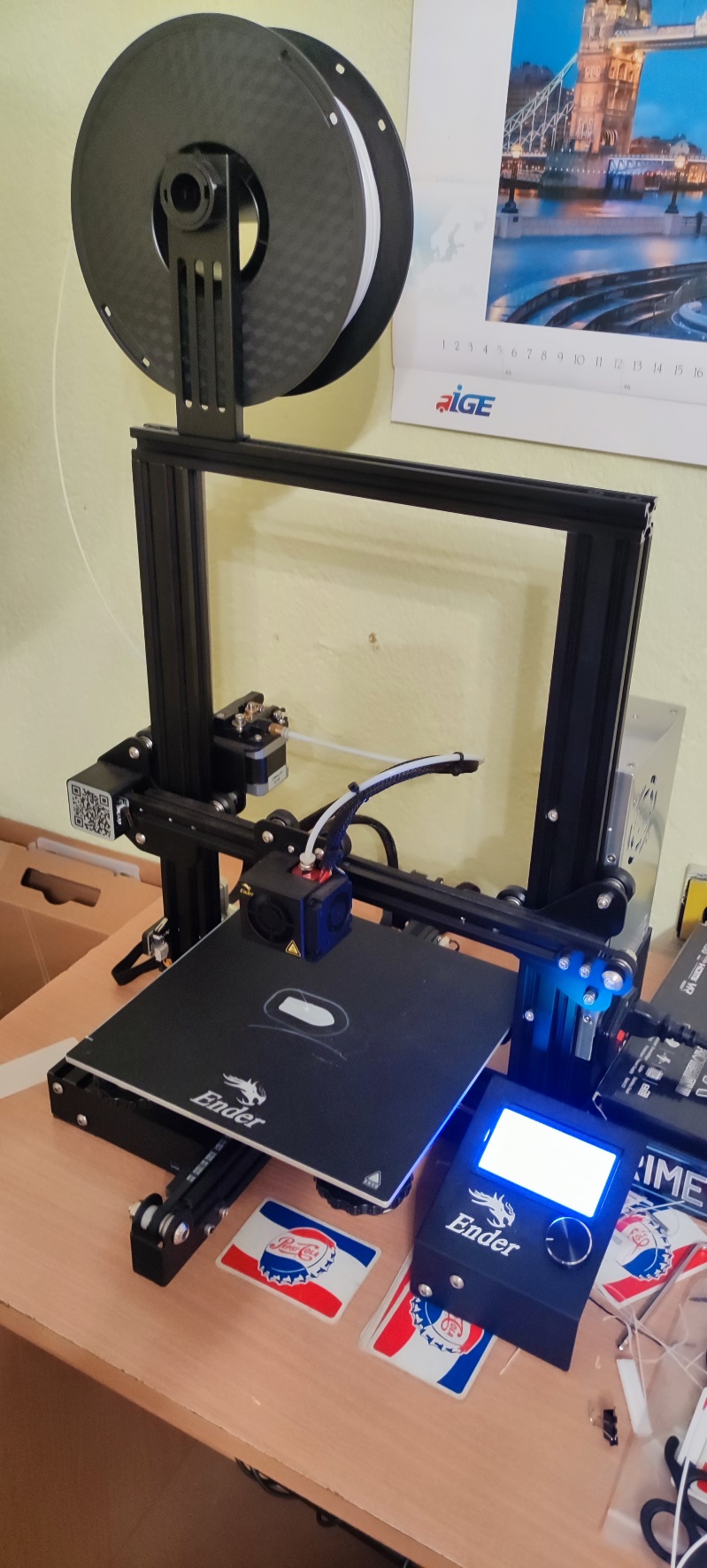
Що се отнася до активните части, вече разгледахме сервомоторите. Следва да споменем камерата, в чийто специфики е ненужно да се спускаме в дълбочина. Само е редно да се отбележи, че камерата е с 5 мегапиксела разделителна способност и сензора поддържа автоматично регулиране на експозицията на образа. Това са двете базови изисквания за по-ефикасна работа на системата. Именно приемлива разделителна способност и яснота на детайлите в образа при лека промяна в светлинните условия.

*Фиг. 5 – Камерата*



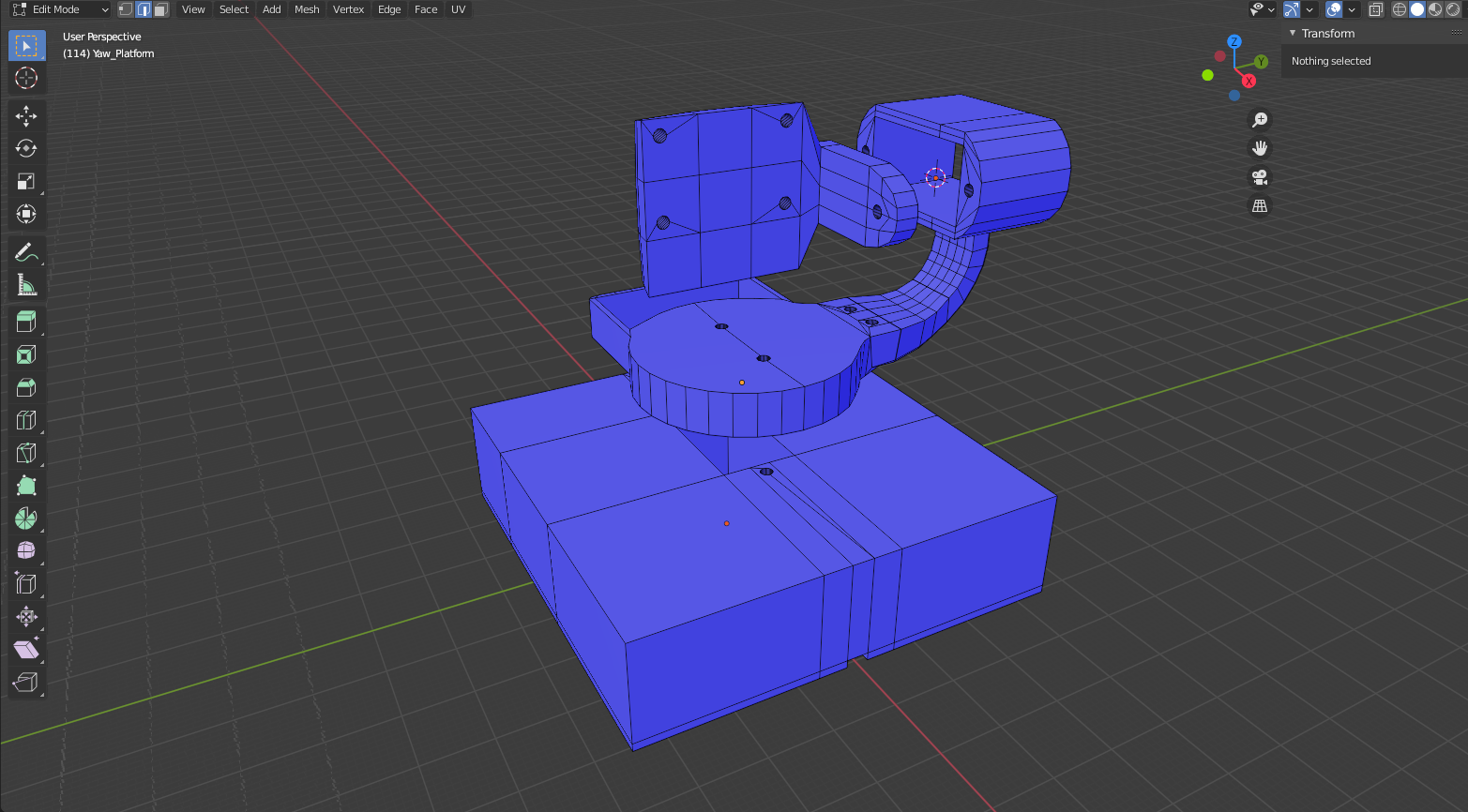
По отношение на пасивните части, предприетия подход за целта на тази задача е 3D принтиране на нужните поддържащи части. Всяка една поддържаща част от механизма е проектирана на CAD програма от мен, след което е принтирана на моя принтер - Creality Ender 3™.

*Фиг. 5 – Използваният 3D Принтер*



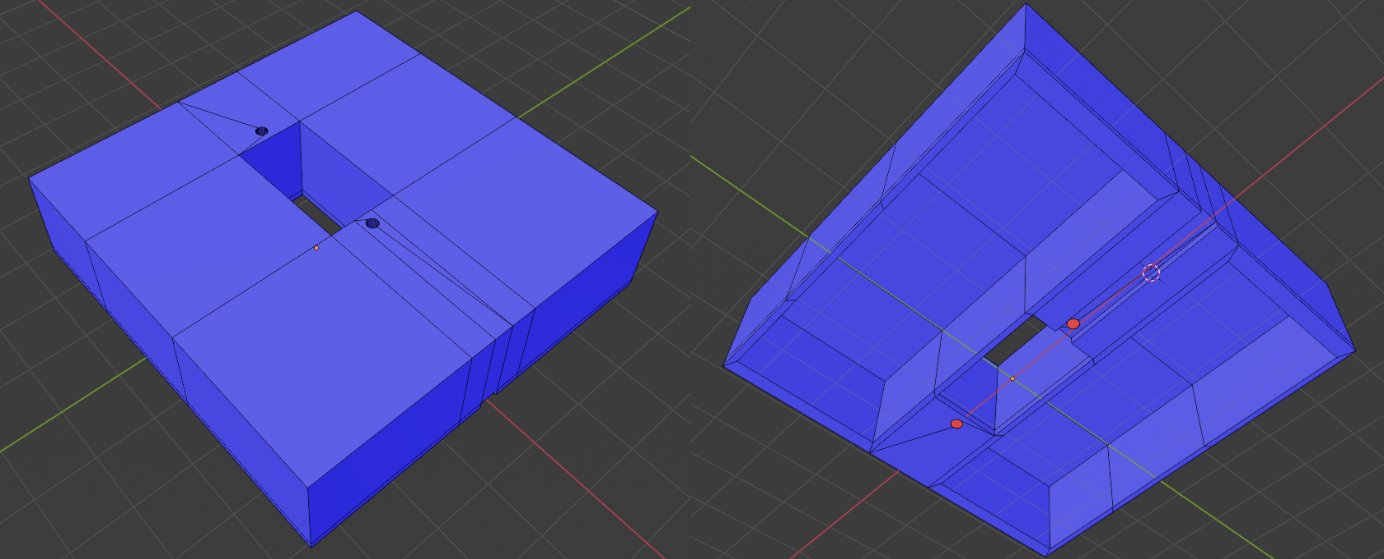
Механизма се състои от 3 поддържащи части. Следва да обсъдим проектирането им без да се спускаме в излишни подробности.

*Фиг. 6 – Поддържащите части в CAD*



Да започнем от основата.

*Фиг. 6.1 – Оглед на основата*

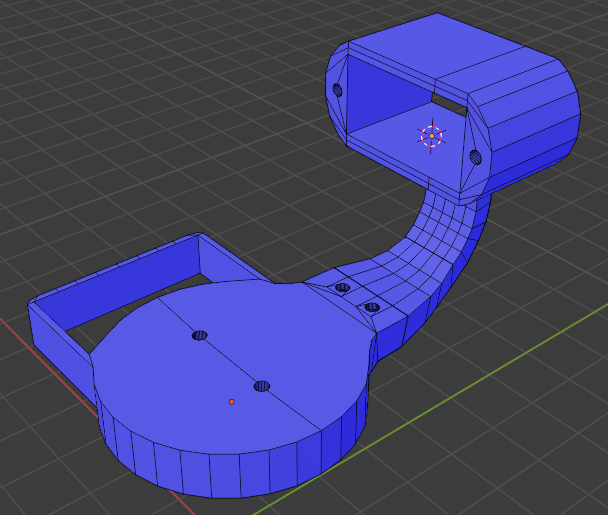
**

С оглед на преносимост и съответствие с пропорциите на активните части, основата е сравнително малка с размерности 70мм x 70мм x 20мм. В средата на основата е проектирана изрезка с размерите на сервото, което ще контролира рисканието**(17)** на камерата. Сервото ляга в този отвор. Тъй като центъра на ротация не съвпада със средата на сервото, то отвора в платформата съответно е изместен с нужното разстояние, така че оста на ротация да попада точно над центъра на платформата.

Допълнително, за да е портативна системата, не бива да се пропуска, че основата ще се поставя без застопоряване върху плоски повърхности. Трябва платформата да е достатъчно тежка, така че като се остави на плоска повърхност, цялата горна част на механизма да може да следва ъгъла на рисканието без да има риск основата да се пързаля на поставената повърхност. За целта е нужно добавяне на изкуствена тежест към платформата. Предприетия подход в случая са две изкуствено добавени кухини в детайла на платформата, които след принтиране се запълват с материал с по-висока плътност. В случая е използван гипс.

Следва да видим средната поддържащата част.

*Фиг. 6.2 – Оглед на средната платформа*

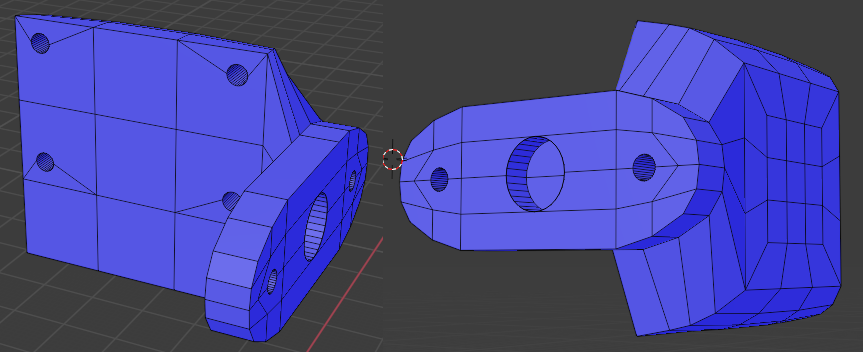


Средната платформа е закрепена по вертикалната ос на ротация и следва рисканието на конструкцията. От лявата страна на частта е проектирано място за монтаж на второто серво, което ще определя тангажа**(17)** на главата и съответно на камерата.

Тъй като камерата има лентов кабел, който ще се спуска до основата, а механизма ще се върти, ще е хубаво да има водач за кабела. Точно това представлява правоъгълната издатина в задната част на платформата.

Следващата и последна основна пасивна част е главата на механизма, на която е монтирана камерата.

*Фиг. 6.3 – Оглед на главата*



Моделът представя равнина за монтаж на камерата и крепежни точки за ръката на страничното серво.

След сглобяване на пасивните и активните части, получаваме целия механизъм с две степени свобода на въртене. Камерата ще може да се насочва подобаващо и механизъма изпълнява поставените функционални изисквания в Глава 2.

**Целият код на представеният проект, както и CAD файловете на частите за механизма е публикуван и наличен с лиценз за отворен код на** (1).

# Софтуер

Ще започнем от устройството на проекта.

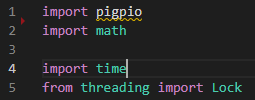
*Фиг. 7 – Устройство на проекта*



Основната папка на проекта е „face\_detection“, тя съдържа файловете с кода на всички функционалности.

Първият файл под вниманието ни ще бъде „**servo\_controller.py**“, който съдържа обекта отговарящ за движението на сервомоторите и огледа.

*Фиг. 8*

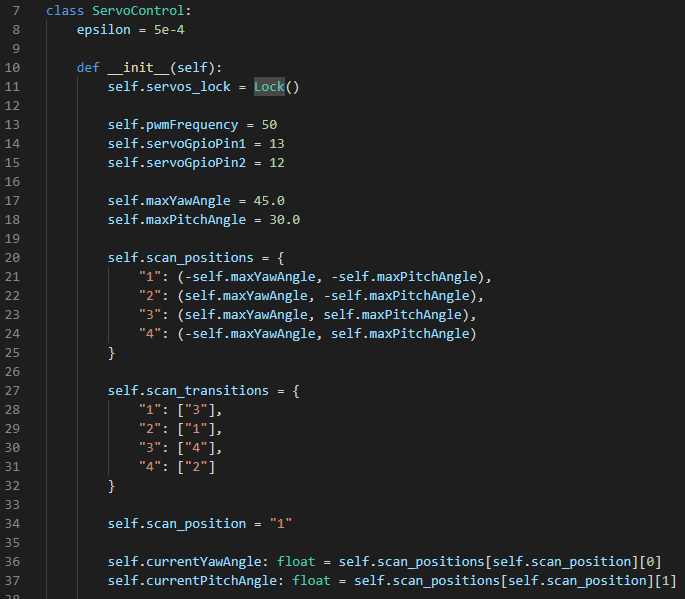


**pigpio** е зависимостта, която ни позволява да контролираме сигнала от GPIO**(18)** пиновете на микроконтролера.

**time** ще ние е нужна зависимост за диференциране на движението на серво моторите.

**threading.Lock** е нужен за заключване на области от кода, които могат да се достъпват от повече от една нишка. Режима на оглед се пуска на отделна нишка от главната на програмата. Главната нишка отговаря за обработка на образа и разпознаването на лицата.

*Фиг. 9*

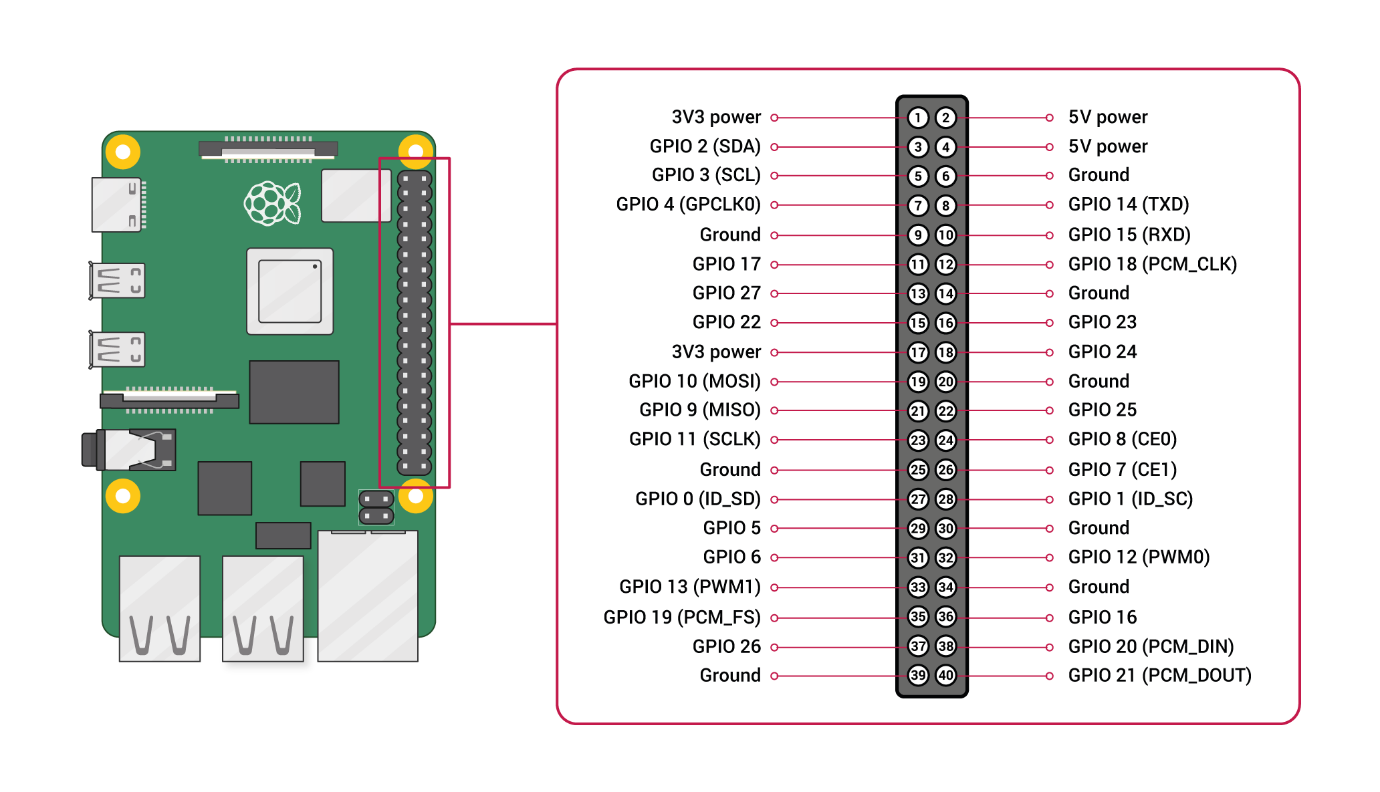


**ServoControl** е основния обект във файла за контрол на сервотата. Нека разгледаме дефинираните му променливи/състояния.

На линия 8 дефинираме „епсилон“ стойност за правилно сравнение на числа с плаваща запетая.

Променливите на редове 13, 14 и 15 дефинират периода на модулирания сигнал както и номерата на използваните GPIO пинове за контрол на двете сервота.

*Фиг. 10 – GPIO пинове на Raspberry Pi 4*

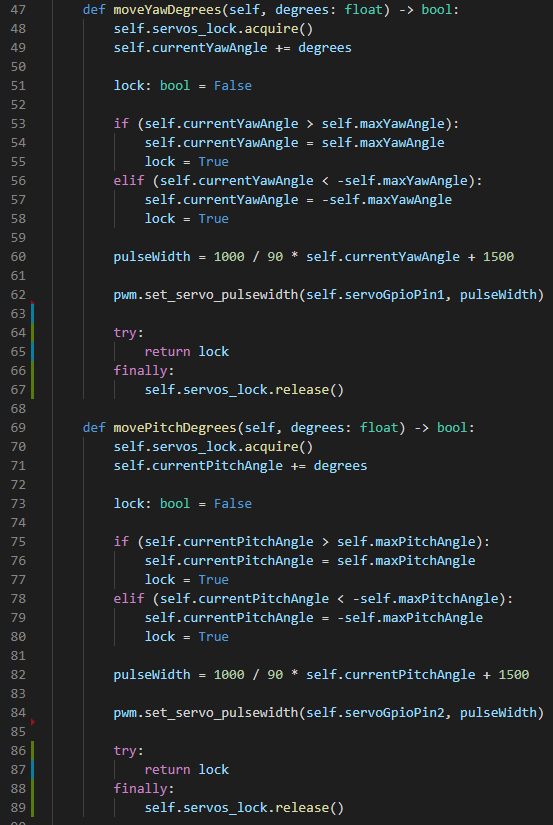


След това се определя максималния абсолютен ъгъл на рисканието и тангажа ако се приеме, че средната позиция в диапазона е нулата. В случая имаме зададени 45 градуса за рисканието и 30 градуса за тангажа на главата.

Между линия 20 и 34 на *Фиг. 9* се дефинират променливите, които ще следят състоянието на панорамния оглед в режим „сканиране“. Отделните позиции на главата са предварително зададени като промяната на състоянията е строго дефинирано чрез „транзиции“, които играят ролята на ребра в свързан граф състоящ се от различните позиции (състояния).

Ред 36 и 37 на *Фиг. 9* са променливи, следящи сегашния ъгъл на рискание и тангаж.

*Фиг. 11*



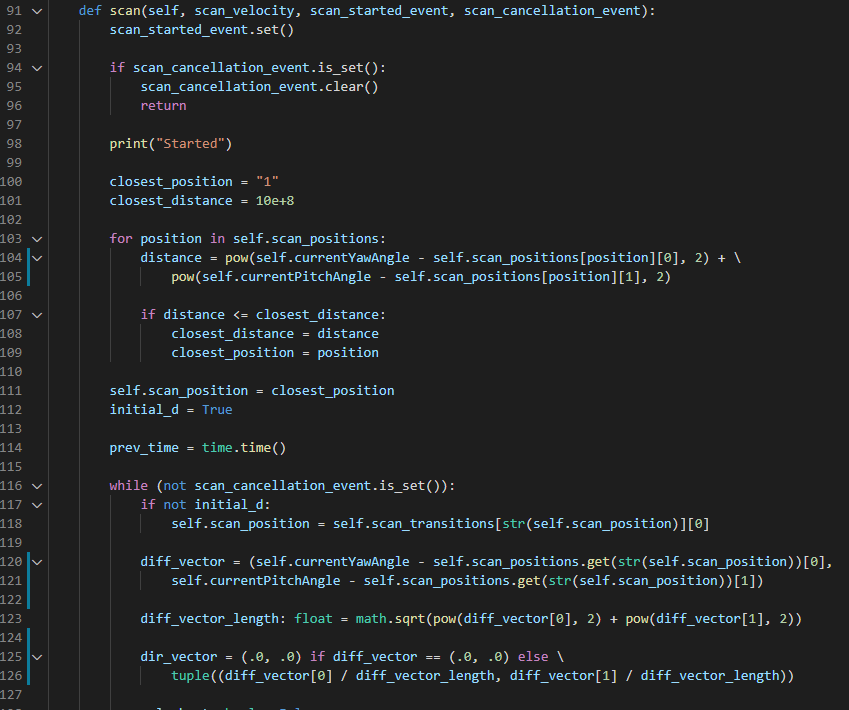
Движението се извършва посредством двете функции **ServoControl.moveYawDegrees(degrees)** и **ServoControl.movePitchDegrees(degrees)**.

Параметъра „degrees“ задава големината на ъгъла на въртене. Ако разгледаме по-подробно първата от тези две функции „**ServoControl.moveYawDegrees(degrees)**“ – тя контролира рисканието на механизма. Ще забележим, че в началото на функцията имаме синхронно заключване по обща ключалка дефинирана в инстанцията на **ServoControl**. Както вече бе споменато, това е нужно тъй като този блок от синхронен код може да бъде извикван от повече от една изпълнителна нишка. Трябва да се гарантира блокиране на всяка нишка в началото на изпълнението на този код ако той вече се изпълнява от друга нишка. Функцията след това проверява дали сервото е достигнало крайна позиция на определения диапазон на ротация. Ако да, вдига флаг **lock** за гранична позиция. След това се изчислява широчината на импулса към сервото в милисекунди като се вземе в предвид очаквания управляващ сигнал описан в Раздел 4.1. Запазва се промяната в сигнала и се изпраща към сервото, при което настъпва движение. Накрая като резултат се връща стойността на флага за гранична позиция.

Функцията **ServoControl.movePitchDegrees(degrees)** отговаря за тангажа на механизма. Тя е абсолютно аналогична на **ServoControl.moveYawDegrees(degrees)**.

Следващата функция от класа е **ServoControl.scan(scan\_velocity, scan\_started\_event, scan\_cancellation\_event)**

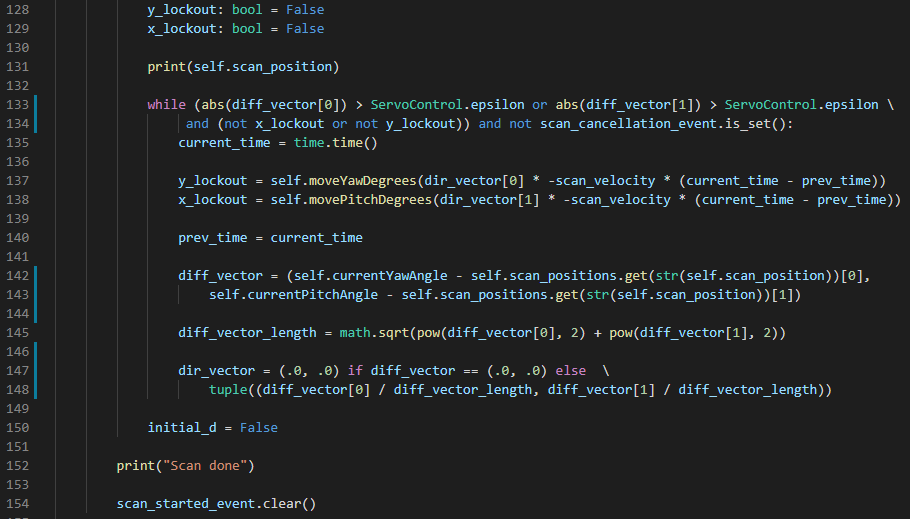
*Фиг. 12*



Параметрите **scan\_started\_event** и **scan\_cancellation\_event** са сигнални примитиви, които се подават от ползвателя на функцията. Целта на **scan\_started\_event** е да съобщи на извикващия, че изпълнението е започнало. В допълнение **scan\_cancellation\_event** има задачата да съобщи, че изпълнението на самото „сканиране“ за лица е приключено. Иначе казано, това би се случило ако програмата е терминирана по очакван начин или лице бъде намерено.

След като сме влезли в изпълнението на този блок от код, първата задача е да намерим най-близката позиция (състояние), към което да насочим камерата. Това се извършва и след това навлизаме в основния цикъл на движение. В него задаваме следващото състояние от графа като цел и изчисляваме вектор към целевата ротация. Допълнително взимаме нормализираната посока към целта.

*Фиг. 13*



След като е зададено следващото състояние от графа, навлизаме във вътрешен цикъл, чийто цел е да гарантира достигането на целевата позиция.

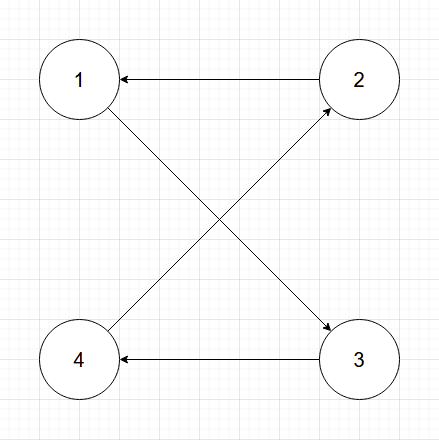
Условието на цикъла е следното: **Ако целта не е достигната (т.е. целевия вектор не е 0) *И* не сме достигнали гранична позиция *И* не е приключено изпълнението, то продължаваме с изпълнението**.

В тялото на цикъла променяме рисканието и тангажа по целевата посока със скоростта **scan\_velocity** диференцирана по времето.

После преизчисляваме целевият вектор за следващото изпълнение, и цикълът се повтаря до достигане на целта или терминиране.

Графът, който е съставен за режима на сканиране е еднопосочен цикличен граф, което гарантира безкрайна цикличност между състоянията.

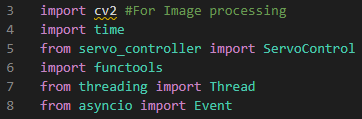
*Фиг. 14 – Граф на маршрута при сканиране (възлите са начертани относително на предвидените позиции една спрямо друга)*



Вече имаме разписана нужната функционалност за движението на механизма.

Минаваме към основния файл на програмата – „**face\_recognition.py**“. В него имаме следните зависимости:

*Фиг. 15 – Зависимости на* ***face\_recognition.py***



**cv2** - Основната библиотека от която се възползваме в проекта. Използва се за обработка на кадрите от видеото на камерата, както и за лицевото разпознаване. Библиотеката предлага и функционалности за създаване на прост графичен потребителски интерфейс, от които ще се възползваме за демонстрацията.

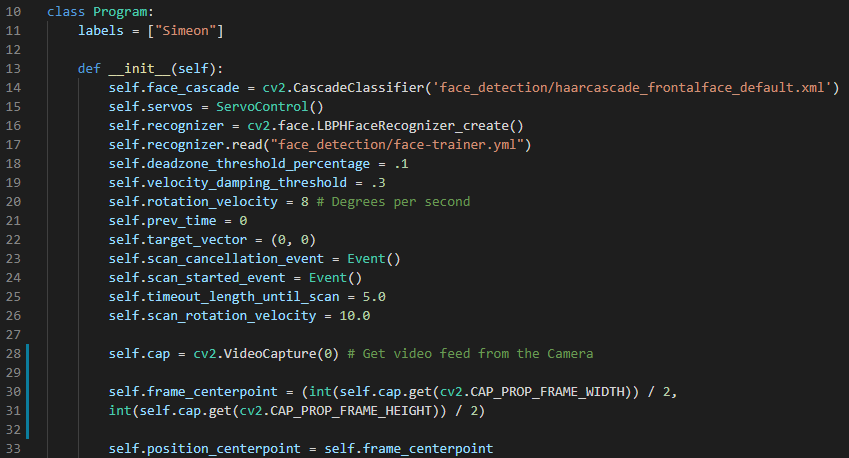
**servo\_controller.ServoControl** – Разписаният от нас обект, чрез който ще контролираме движението на камерата.

**functools** – Тук са дефинирани полезни функции за боравене с колекции.

**threading.Тhread** – С този обект може да създаваме нишки за паралелно изпълнение.

**asyncio.Event** – Сигнален примитив, който ще е нужен за паралелно изпълнение.

*Фиг. 16*



Нека разгледаме дефинираните основни променливи на програмата.

**face\_cascade** – Създаваме каскаден класификатор**(19)** с цел откриване на конкретен обект в изображение. В нашия случай му подаваме трениран модел за ***Хаар наподобяващи черти на лица***. С други думи, този класификатор ще бъде използван за разпознаване на лица. За повече информация виж Раздел 5.1.

**servos** – Инстанция на обекта ни за управление на камерата.

**recognizer** – Тук създаваме лицев идентификатор базиран на ***алгоритъм, възползващ се от хистограми на локални двоични черти***, за да намира подобия при сравнение на изображения с вече изграден модел от тренировъчни изображения. В нашия случай изображенията са лица. За повече информация виж Раздел 5.2

**deadzone\_threshold\_percentage** – Процент разстояние на лицата до централната точка на екрана, след което камерата ще спре да следи. Това се прави с цел да остави прозорец, в което леки движения на лицата няма да провокират движение от страна на камерата.

**velocity\_damping\_threshold** – Процент разстояние на лицата от центъра на екрана, след което камерата постепенно ще намалява скоростта на въртене, за по-плавно центриране в кадър.

**rotation\_velocity** – Ъгловата скорост на механизма (градуси в секунда).

**target\_vector** – Вектор сочещ от центъра на екрана към лицето.

**scan\_cancellation\_event** – Сигнален примитив за прекъсване на сканиране (ако тече такова)

**scan\_started\_event** – Сигнален примитив за отбелязване на започнало сканиране на друга нишка.

**timeout\_length\_until\_scan** – Периода от време, който трябва да измине без намерено лице, преди механизма да влезне в режим на сканиране.

**scan\_rotation\_velocity** – Ъгловата скорост на механизма в режим сканиране (градуси в секунда).

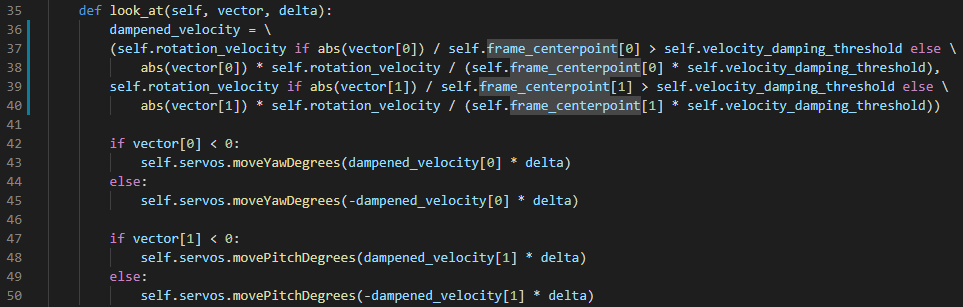
**cap** – Инстанция на обект от OpenCV, който ни предоставя кадрите от камерата в реално време.

**frame\_centerpoint** – Централната точка на картината.

**position\_centerpoint** – Средната точка между всички намерени лица в кадър.

Действието за поглеждане на определена точка има някои конкретики, които трябва да се вземат на предвид преди директното извикване на функциите за въртене на сервотата. За целта е написана отделна функция **look\_at(vector, delta)**

*Фиг. 17*



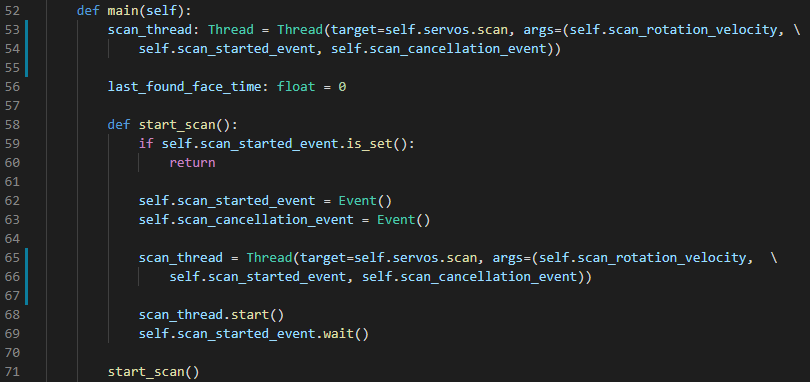
Задачата на тази функция е да определи посоката на въртене и да приложи функционална трансформация върху скоростта на въртене. До определена част от пътя, скоростта е максималната зададена скорост на въртене. След преминаване на зададената граница , скоростта става линейно обратнопропорционална спрямо дистанцията до вече подминатата граница. Математически записана, функцията за скоростта е следната:

Съответно пътят, който ще измине устройството се получава чрез постъпкова интеграция на , която се осигурява от главния цикъл на програмата:

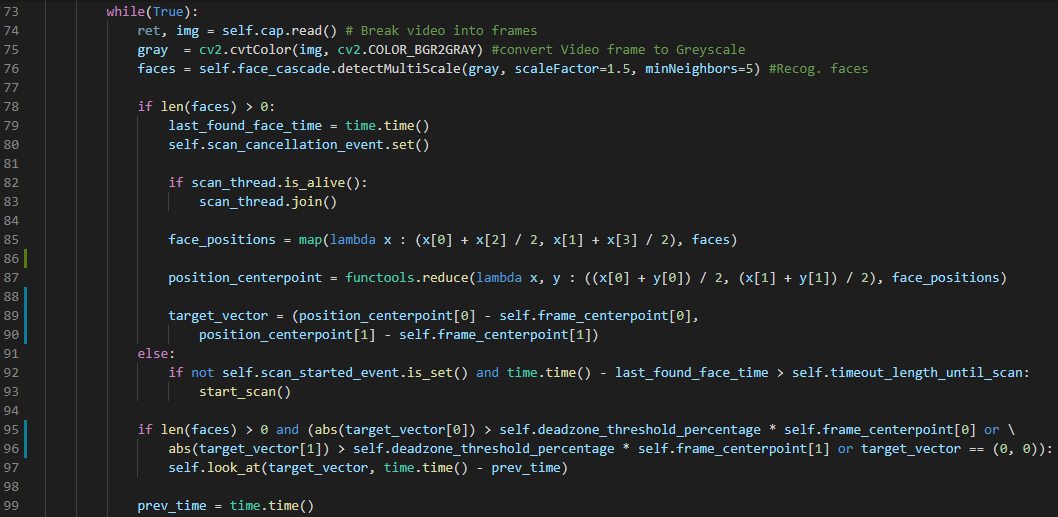
Тъй като по условие ние се стремим да намалим пътя до целта, позицията на сервотата се мени в обратната посока на оставащия път:

Това означава, че в рамките на тази задача, множителя използван за потушаване на променливата се държи по точно очаквания начин, а именно:

*Фиг. 18*

Главната функция на програмата започва като създава обект за нишката, която ще бъде изпълнена паралелно когато устройството влезе в режим сканиране. След това заделяме кода за пускането на нишката в отделна локална функция с цел избягване на повторяемост. Веднага извикваме тази функция, за да започне паралелно сканирането в началото на изпълнението.

*Фиг. 19*



Сега паралелно със сканирането навлизаме в основния цикъл на програмата. Първо се взима текущия кадър от видео потока. След това изображението се записва и в монохромен формат, който е нужен за вход на класификатора за лице. Използваме класификатора дефиниран в програмата за намиране на лицата в монохромния образ и взимаме резултата като региони с лица записани в колекция „**faces**“.

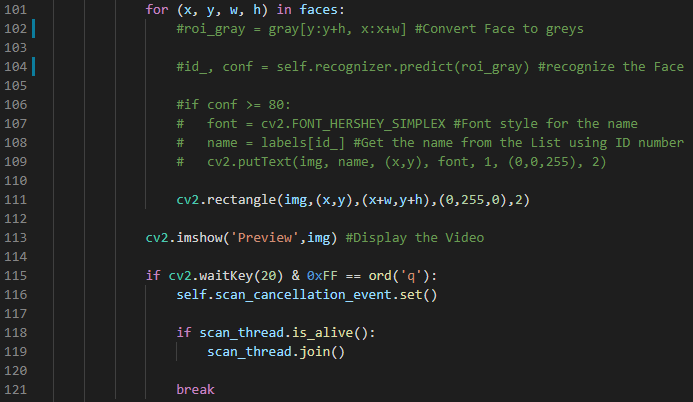
Следва проверката на гореспоменатата колекция с лица.

Ако **има** намерени лица (колекцията не е празна), то трябва да сигнализираме на паралелната нишка, която изпълнява кода за сканирането (ако има такава работеща) да приключи изпълнението. После взимаме позициите на лицата и изчисляваме средната точка между намерените лица. Тази позиция става основна точка на интерес. Камерата трябва да започне да се придвижва към нея. За целта се изчислява вектора към тази позиция.

Ако **няма** намерени лица се проверява дали в момента тече сканиране на отделна нишка. Ако не тече сканиране и е изминало повече от определеното време без намерено лице, то се пуска отделна нишка за сканиране и устройството отново минава в този режим.

При намерени лица, ако точката на интерес е извън определения централен регион, то се извиква функцията **look\_at(vector, delta)**, която както вече разгледахме, интегрира функцията на пътя, за да се придвижи към целта. Съответно ѝ се подава **delta** като параметър, който реално представлява изминалото време от предишното ѝ сумиране. Резултата е ротация на механизма в ход към точката на интерес.

*Фиг. 20*



Проверяваме колекцията от лица поелементно. Тук имаме избора да решим дали имаме възможността да идентифицираме определено лице за което сме тренирали лицевия идентификатор. В този случай, за демонстрационни цели кода за идентифициране на лицата е коментиран, защото ни липсват съществени тренировъчни данни и това не основна цел на тази работа. За пълнота обаче следва да обсъдим и коментирания код, защото програмата има възможност да идентифицира лица при налични тренировъчни данни.

За всяко лице се взима региона, в който се намира, от монохромния образ. След това този регион състоящ се от лицето се подава като вход на лицевия ни идентификатор. Ако има база за сравнение, той ще предвиди подобие на лицето със стойностите на изградените двоични хистограми за определен човек. При съвпадение с достатъчно голяма увереност, над лицето се изписва текст с името на предположения човек от тренировъчната база от лица. Тази функционалност, въпреки че е коментирана от кода, действа при добре сформирана тренировъчна база от образи и ще е част от демонстрацията. За целта на демонстрацията съм разписал допълнителен скрипт, който е част от проекта под името „**face\_data\_accumulator.py**“. Умисъла с който е написан този скрипт е да подпомогне изграждането на база от тренировъчни образи с помощта на камерата на устройството. Допълнително ще се видят в пълнота тези възможности при демонстрацията на устройството.

Сега нека продължим по същество с основната задача. Тъй като лице (идентифицирано или не такова) е намерено, ние рисуваме границите на региона, в който се намира. След като направим горе-споменатото за всички намерени лица, представяме обработения образ и основния цикъл се превърта наново за следващия кадър от видео потока ако няма потребителски сигнал за изход от програмата.

Ако има сигнал за потребителски изход, приключваме изпълнение на работещите паралелни нишки (ако има такива) и излизаме от главния цикъл на програмата.

*Фиг. 21*



Разчистват се заделените ресурси и програмата приключва.

Глава 5 – Принцип на работа

По отношение на работата на устройството има два ключови момента по време на изпълнение на софтуера, които са в центъра на функционалността. Единият е отличаването на човешко лице в образ, а другият е идентифицирането на съответното лице. По отношение на първия проблем е полезно да разгледаме какво всъщност представлява **Каскаден класификатор базиран на Хаар черти**. За втория проблем ще се впуснем в естеството на един конкретен алгоритъм, най-често използван в компютърното зрение**(20)** - **Алгоритъм за сравнение използващ Хистограми на Локални Двоични Черти**. Да започнем поетапно.

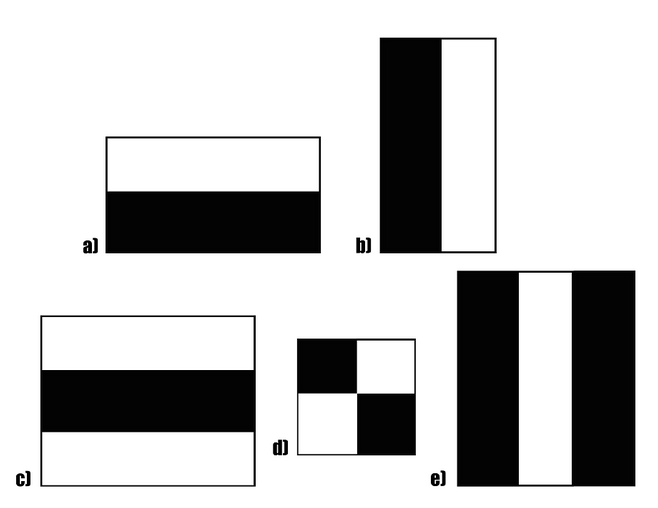
# Каскаден класификатор базиран на Хаар черти

Та как точно работи алгоритмично откриването на едно човешко лице в изображение/кадър от видео? Алгоритъма използва характерни черти на изображения, описващи ръбове и линии, които са предложени от Паул Виола и Майкъл Джоунс в тяхната взаимна научна теза публикувана под името Rapid Object Detection using a Boosted Cascade of Simple Features (2001) [1]. На алгоритъма се подават много положителни снимки съдържащи лица както и много отрицателни снимки без лица с цел обучение. Резултатния модел е отворен за ползване и може да се снабдим с него от официалната GitHub репозитория на OpenCV: <https://github.com/opencv/opencv/tree/master/data/haarcascades>

Моделите са налични под формата на .XML файлове съдържащи векторите от тегла и данни, които може да бъдат прочетени от дефинирани функции в OpenCV. Моделът, който е използван за нашите цели е файл под името „**haarcascade\_frontalface\_default.xml**“ и се намира във файловете на проекта, както може да бъде видян във *Фиг. 7*.

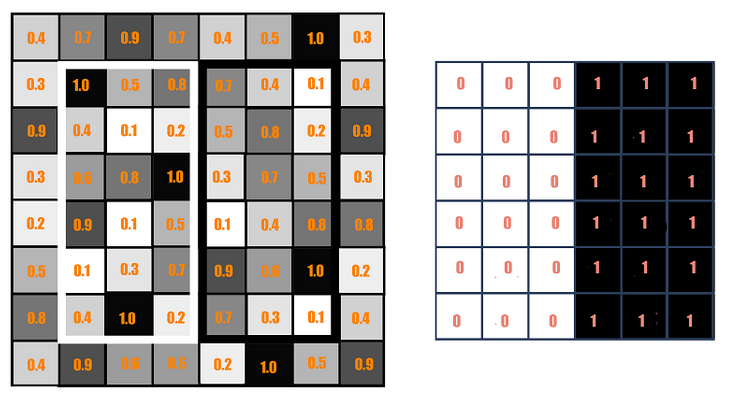
Нека разгледаме първия принос към проучването[1] , а именно Хаар чертите показани по-долу.

*Фиг. 22 – Хаар черти използвани в оригиналната публикация*[1]

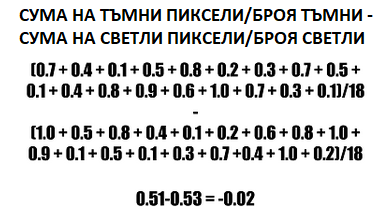


Тези черти на изображението ни позволяват да открием лесно ръбове и прави в образа или да изберем области, в които има рязка промяна на интензитетите на пикселите.

Фиг. 23 – Практически поглед на проверка за Хаар черта



Фиг. 23.1 – Резултат от проверката



Да обърнем внимание на *Фиг. 23.* Правоъгълника отляво представлява пример за изображение с пиксели със стойности от 0,0 до 1,0. Правоъгълника отдясно представлява Хаар ядро, в което всички светли пиксели са отляво, а всички тъмни – отдясно. Хаар изчислението, което се вижда във *Фиг. 23.1* се случва като се вземе разликата между средната стойност от тъмния регион и средната стойност от светлия регион. Ако разликата е близко до 1, то има ръб открит от Хаар чертата. В нашия пример стойността е далече от 1, тоест ръб не е открит.

Всяка една Хаар черта се състои от светли региони репрезентирани от 0 и тъмни региони репрезентирани от 1. Тяхната цел е да откриват специфични характерни черти в дадено изображение. Например ръб, права или каквато и да е структура в образ, където се наблюдава рязка промяна на интензитети. Например в примера даден по-горе, представената Хаар черта търси вертикален ръб с по-тъмни пиксели отдясно и по-светли пиксели от ляво.

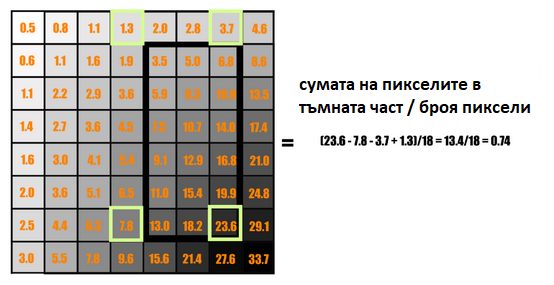
Сега това беше една конкретна Хаар черта, разделяща вертикален ръб, но има много други видове Хаар черти, които биха открили ръбове и в други посоки както и много други по-особени структури. За да се направи това, Хаар чертата трябва да премине поетапно през цялото изображение. По конкретно, чертата трябва да премине през всеки пиксел на изображението. Както и трябва да се използват всички възможни големини на Хаар чертата, които биха се побрали в региона за проверка. В зависимост от чертата, която е нужна, те могат да се класифицират в три категории. Първата група се състои от *черти с два правоъгълни региона* и те се използват за намиране на ръбове във вертикална или хоризонтална посока. Втората група се състои от *черти с* *три правоъгълни региона*, които се използват за намиране на светъл регион, заобиколен от тъмен регион и обратното. Третата група състояща се от *черти с четири правоъгълни региона* са отговорни за откриване на промяна в интензитет по диагонали.

Както можем да видим от примера даден във *Фиг. 23*, проверката на изображение спрямо Хаар черта би включвала много математически операции. Само в дадения пример имахме 18 сумарни операции (за правоъгълник обграждащ 18 пиксела). Сега да си представим това за цялото изображение за всеки възможен размер на чертата. Това е много щура операция, даже и за машина с висока производителност.

За да се преборят с този проблем, проучващите предлагат решение под формата на така нареченото „Интегрално изображение“, върху което да приложат същата операция. Интегралното изображение се пресмята като за всеки пиксел се поставя стойност, която е сумата на стойностите на всички пиксели над и отляво на него. По този начин, като се вземе правоъгълен регион, сумата на стойността на интензитетите се свежда до сумарна операция на само 4 константи. Това е възможно, тъй като пикселите в 4-та ъгъла на който и да е взет регион съдържат сумирания интензитет на пикселите до този ъгъл. Долният десен и горният ляв интензитет се сумират, а долният ляв и горният десен се изваждат от тази сума.

Фиг. 24 – Проверка за Хаар черта върху Интегрално изображение

(много по-малко сумарни операции)



До тук разгледахме същността на проверките, които се извършват върху образите в оригиналното проучване по Хаар Каскади. Сега е време да разгледаме някои подробности на имплементацията.

Това което до тук разбрахме е, че има множество от черти, които биха открили определени форми на лица като например вежди, разделението между очите или устни и т.н. Но в началото, групата от търсените черти не е била лимитирана само до това. Множеството от използваните черти се е състояло от около 180 000 броя черти, които са били селективно намалени до 6000 на по-късен етап.

Голяма част от тези 180 000 черти няма да работят добре или ще са неадекватни що се отнася до лицеви черти, защото ще са твърде случайни за намиране на каквото и да било. За това е трябвало да се разработи технология за селективен подбор на черти. Идеята е да се избере от тях подмножество, което не само ще се представи по-добре от другите черти, но и ще елиминира ненужните. Тук са използвали техника за така нареченото „буустване“ (*на английски* – **Boosting Technique(21)**) на име **AdaBoost(22)**, при което всяка от тези 180 000 черти са приложени към изображения по отделно, за да създадат така наречените „Слабо учащи се“ (*на английски* – **Weak Learners(23)**). Някои от тях изкарват ниска честота на грешки докато разграничават положителните изображения от отрицателните по-добре от другите. Тези „слабо учащи се“ множества от черти са разработени по такъв начин, че да грешат в класификацията възможно най-малко. Представят се по-добре от случайно отгатване. С тази техника, крайното множество от черти е редуцирано до 6000.

Сега минаваме на каскадната част от алгоритъма. Това подмножество от всичките 6000 черти ще се пусне за сравнение върху тренировъчните изображения, за да реши дали има черти от лица във тях или няма. Авторите на алгоритъма са взели като стандартен размер на прозореца за проверка - 24х24 пиксела. Отново, проверката спрямо всички тези черти дори и в този размер е много тежка задача.

За да опростят това предлагат друга техника наречена **Внимателна Каскада** (*на английски* – **Attentional Cascade**). Идеята зад нея е, че не всички черти трябва да се проверяват в избрания регион за проверка. Ако някоя черта се провали в проверката в избрания регион, то може да кажем, че там няма лице и да изберем следващ регион, пропускайки сегашния.

Чертите се прилагат към изображенията на етапи. Етапите в началото съдържат само по-прости черти в сравнение с чертите в по-горните етапи, които са сложни – достатъчно сложни да открият всички по-специфични и особени дребни детайли в едно лице. Ако първият етап не открие нищо в региона, тогава този регион се отхвърля от процеса на проверка и се преминава на следващия. По този начин ще бъде спестено много време тъй като по-голямата част от безинтересните региони ще бъдат отхвърлени още в началото.

Етапите описани в оригиналното проучване на Виола и Джоунс са 38 на брой за около 6000 различни черти. Броя заделени черти за първите няколко етапа са разпределени съответно в групи с по 1, 25, 25 и 50 черти. И бройката се увеличава последователно за следващи етапи. Първите етапи премахват повечето от участъците в образа на които не е открито лице, докато по-късните етапи се фокусират да намалят грешката.

# Хистограми на Локални Двоични Черти – Алгоритъм за сравнение

**Локална Двоична Черта (ЛДЧ)**,*на английски* – **Local Binary Pattern (LBP)** е прост, но в същото време доста ефикасен текстурен**(24)** оператор, който бележи пикселите на изображение като сравнява по гранична стойност съседството на всеки пиксел и подразбира резултата като двоично число.

Този оператор е описан за пръв път през 1994 и от тогава е разпознат като много мощно средство за класификация на текстури.

Използвайки **ЛДЧ** в комбинация с хистограми може да представим изображения на лица като прост вектор от данни.

Тъй като **ЛДЧ** е визуален дескриптор той може да се използва и с цел идентифициране на лица. Следва да опишем точно как се случва това.

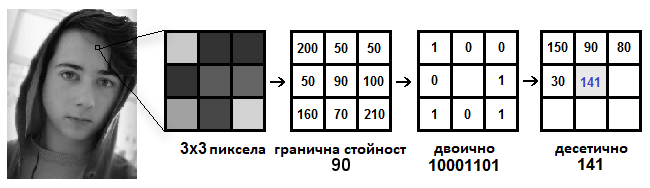
Основните стъпки за сравнителния алгоритъм са следните:

1. ***Параметри***
   1. **Радиус** – Радиусът се използва за построяване на кръгла локална двоична черта и представя радиусът около централния пиксел. По принцип се задава на стойност 1.
   2. **Съседи** – Броя на пробните точки за построяване на кръгла локална двоична черта. Колкото повече точки се вземат, толкова по-скъпо става изчислението. По принцип се използват 8 пробни точки.
   3. **X на мрежата** – Броя на клетките/подразделението на мрежата в хоризонтална посока.
   4. **Y на мрежата** – Броя на клетките/подразделението на мрежата във вертикална посока.

Колкото повече клетки има в мрежата, толкова по-голяма е размерността на резултатния вектор на модела.

1. ***Обучение на алгоритъма***: Първо трябва да се обучи алгоритъма. За да го направим ще трябва да се използва база от образи на лица на хора, които искаме да разпознаем. Трябва също така да прикачим уникален идентификатор (например името на човека или номер) за всяка снимка, за да може алгоритъма да използва тази информация за разпознаване на входна снимка и изкарване на резултат. Изображения на един и същи човек трябва да носят идентичен идентификатор. След като е конструирана базата от данни за обучение може да разгледаме стъпките на самото изчисление. Тренировъчните данни в тази работа се съхраняват в директорията „**faces**“, като името на всяка поддиректория представлява идентификатора за съответния човек.
2. ***Прилагане на ЛДЧ операцията:*** Първата изчислителна стъпка на ЛДЧ-хистограмния алгоритъм е да създадем ново изображение-посредник, което описва оригиналното изображение по по-желателен начин като подчертава лицевите характеристики. За да го постигне, алгоритъма използва концепцията на плъзгащ се прозорец, който е базиран на параметрите **радиус** и **съседи**.

*Фиг. 25 – Описание на процедурата*



Нека опишем какво точно се случва в по-горното изображение.

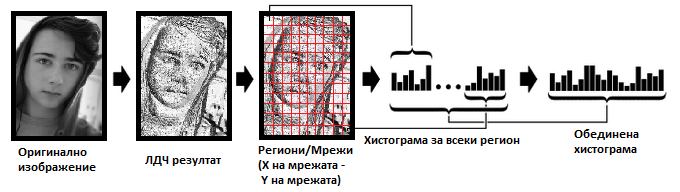
* Представено ни е монохромно изображение на лице.
* Може да вземем част от лицето като прозорец от 3 на 3 пиксела.
* Може да се репрезентира като 3х3 матрица съдържаща интензитета на всеки пиксел (От 0 до 255).
* Тогава трябва да вземем централната стойност на матрицата като гранична стойност.
* Тази стойност ще бъде използвана да се дефинират нови стойности от 8-та съседа.
* За всеки от съседите на централната (граничната) стойност, ще зададем нова двоична стойност. Задаваме 1 за стойностите по-големи или равни на граничната и 0 за стойностите по-малки от граничната.
* Вече матрицата съдържа само двоични стойности (игнорирайки централната). Трябва да обединим всяка двоична стойност от отделните позиции на матрицата ред по ред за да получим нова двоична стойност (напр. 10001101). *Бележка*: Може да се използва и друга поредност за взимане на двоичните стойности, например изреждането им обратно на часовниковата стрелка. В крайна сметка резултата ще е същия при консистентен подход.
* След това обръщаме тази двоична стойност в десетична и я задаваме като централната стойност.
* На края на процедурата, която изпълнява това за всеки пиксел от образа, получаваме ново изображение, което по-добре представя характеристиките на оригиналното изображение.
* Уместно е да се спомене, че ЛДЧ процедурата бе разширена да използва различен брой радиуси и съседи, нарича се **Кръглова ЛДЧ**.

*Фиг. 26 – Кръглова ЛДЧ*

При кръглова ЛДЧ се използва билинейна интерполация**(25)** за взимане на цветовете от 4-та най-близки пиксела като стойността на пробната точка лежаща на описаната окръжност.

1. **Извеждане на хистограмите**: Сега използвайки изображението генерирано от миналата стъпка, може да използваме параметрите за Х на мрежата и Y на мрежата да разделим образа на отделни клетки, както можем да видим на фигурата по-долу.

*Фиг. 27 – Разделяне на ЛДЧ образа на отделни клетки и извеждане на хистограма за всяка клетка*



* Тъй като имаме монохромен образ, всички хистограми (за всяка клетка) ще съдържат само 256 позиции (От 0 до 255), които представляват интензитетите за всеки пиксел от съответната клетка.
* Тогава трябва да обединим всяка хистограма, за да сформираме една нова по-голяма такава. Ако например имаме 8 на 8 мрежа от клетки, то ние ще имаме 8х8х256 = 16 384 позиции в крайната обединена хистограма. Тя репрезентира характеристиките на оригиналния образ.

С това общо взето приключва изпълнението на ЛДЧ-хистограмния алгоритъм.

На нас обаче ни остава още една стъпка за обсъждане.

1. **Изпълнение на лицевата идентификация**: В тази стъпка алгоритъма е вече обучен. Всяка създадена хистограма е използвана да представя всяка от снимките в тренировъчната база данни. И така, като предоставим чисто ново изображение за вход, изпълняваме и за него същите стъпки за сформиране на хистограмата му.

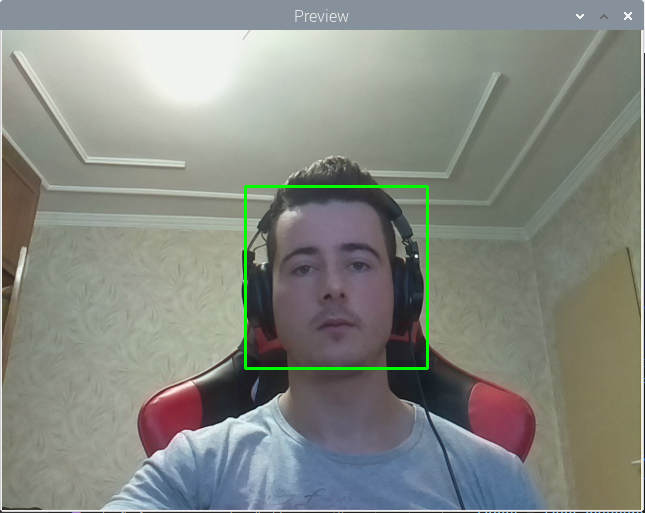
* За да намерим изображение, което съвпада с входната снимка ние само трябва да сравним двете хистограми и да върнем снимката с най-близка такава.
* Може да използваме най-различни подходи за сравнението на стойностите от хистограмите, например: евклидова дистанция, хи-квадрат, абсолютна стойност и т.н; В нашия пример може да използваме евклидова дистанция. Тя се базира на следната формула:
* И така изходния резултат от алгоритъма е идентификатора на съответното изображение с най-близката хистограма. Алгоритъма също би следвало да върне и изчислената дистанция, която може да се използва като един вид мярка на „увереност“. *Забележка*: Не трябва да се подвеждаме от името „увереност“ тъй като реално, колкото по-ниска е дистанцията, толкова по-близки са двете хистограми.
* После може да зададем гранична стойност за „увереността“, за да определим дали алгоритъма правилно е установил лицето. Ако увереността е под зададената граница, считаме лицето за коректно идентифицирано.

Глава 6 – Демонстрация

Демонстрацията в този документ ще покаже характерната обработка на образа, който вижда камерата.

Първо нека видим какво излиза на екрана без да идентифицираме каквито и да е лица.

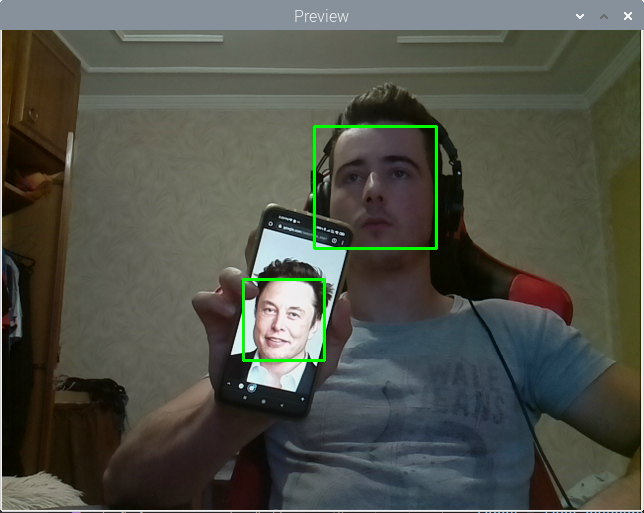
*Фиг. 27 – Едно лице без идентификация*



Както виждаме, устройството открива лицето ми в образа и ме поглежда директно, центрирайки лицето ми в кадър.

Сега нека видим какво се случва ако добавим и второ лице в ситуацията.

*Фиг. 28 – Две лица без идентификация*



Двете лица са открити успешно и камерата поглежда средната точка между позициите им, побирайки и двете лица в кадъра.

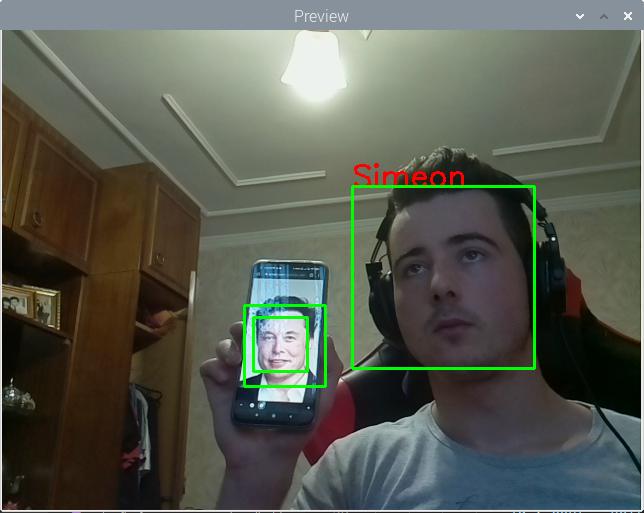
Сега е време да се възползваме от двата допълнително разписани файла в проекта: „**face\_data\_accumulator.py**“ и „**face\_trainer.py**“.

Както вече бе споменато набързо в Раздел 4.2, **face\_data\_accumulator.py** е скрипт разписан с идеята да събере тренировъчни данни на момента, за лице намиращо се пред устройството. Това, което реално прави програмата в този файл е използва същия метод обсъден в Раздел 5.1, за да намери лице в образа пред камерата. След което взима съответния кадър, в който е намерено лице и го запазва във „**faces/[името на човека който е въведен за тренировка]**“. Това се случва с готовия обект, предоставен от OpenCV и съответно кода е кратък и обсъждането му е тривиално. Ако желаем може вместо това да качим готови снимки като тренировъчни данни в директория създадена по гореспоменатата схема, за да добавим лице за разпознаване, което не е достъпно пред камерата.

След като съберем снимки на определено лице от устройството (или качим тренировъчни снимки), трябва да сформираме матрици съдържащи информацията от техните **ЛДЧ хистограми**. Именно това е целта на скрипта във файл **face\_trainer.py**. Използвайки метода описан в Раздел 5.2, OpenCV функцията, която използваме създава една обединена хистограма на лицата намерени във всяка една от тези снимки за конкретни хора, намиращи се в директория „**faces**“. Тези обединени хистограми (заедно с идентификаторите на хората, които представляват) после се записват във файл на име „**face-trainer.yml**“. Този файл се подава на лицево разпознаващия **ЛДЧ-хистограмен** обект на OpenCV, който се използва в основната ни програма и сравнява лицата по време на изпълнение.

Създадох тренировъчни данни за лицето ми и после сформирах техните хистограми. Нека видим как се представя идентификацията на лица с подадените данни.

*Фиг. 29 – Две лица, само с тренировъчни данни за моето лице*

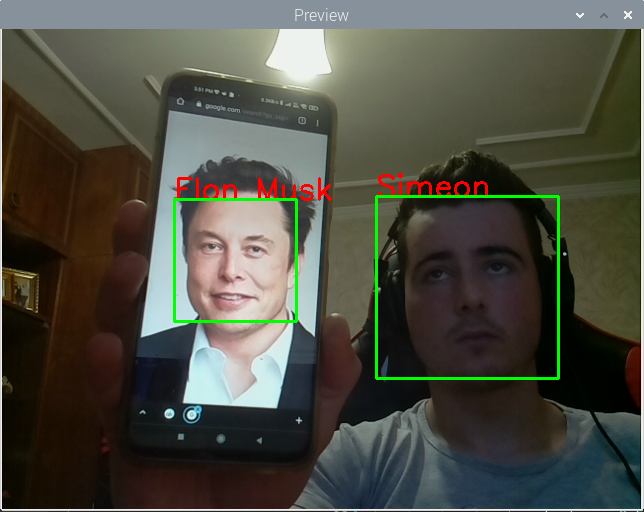


Както виждаме имаме открити повече лица, но моето вече показва името ми над очертания регион. Изглежда моето лице е най-близкото от показаните до хистограмния профил изграден от моите снимки.

Като последен тест нека добавим ръчно тренировъчни снимки за някоя известна личност – например Илон Мъск. Ще ги свалим на готово от тук:  
<https://www.kaggle.com/anku5hk/5-faces-dataset>  
След което ги слагаме във „**faces\Elon Musk**“ директорията. После сформираме тренировъчните стойности наново.

Нека видим резултата от добавените стойности за второто лице.

*Фиг. 30 – Две лица, с тренировъчни данни и за двете*



Както виждаме и двете лица са разпознати коректно. Трябва да се спомене, че в зависимост от интензитета на различни сенки на ключови места, може две подобни лица да се доближават в профила си прекалено много. Това ще доведе до грешно определено лице. Особено в случая когато в кадър бъде заснето само едно лице. За жалост, това е слабост на използваните методи.

Резултатите може да се подобрят с избиране на различна по-подобаваща гранична стойност за дистанцията („увереността“) между двата профила. Желателно е и използването на по-подобаващи или актуални тренировъчни данни.

С това приключва демонстрацията на устройството.

Глава 7 – Заключение

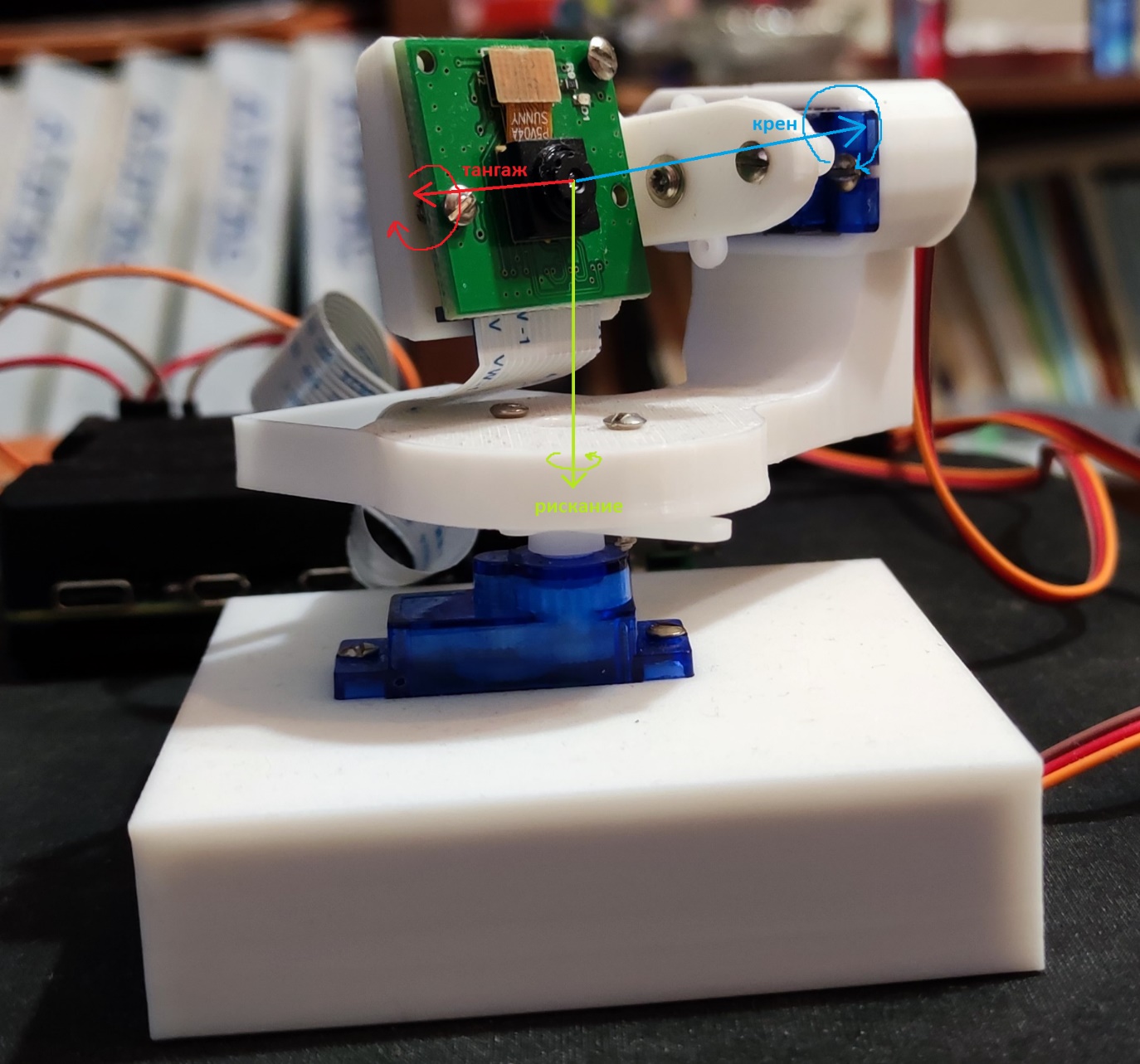
Благодарение на екстензивните проучвания в сферата на компютърното зрение в последните десетилетия и чудесни библиотеки със свободен достъп, в момента технологиите за разпознаване на лица са по-известна тема от всякога. Достъпността на библиотеки като OpenCV ни позволяват да разработваме всякакви проекти свързани с компютърно зрение и машинно обучение, както и да адаптираме възможностите на сегашните технологии към нашите конкретни изисквания.

В тази работа демонстрирахме едно от многото потенциални приложения включващи машинно обучение, а именно - лицевото разпознаване. Ефективно разгледахме използваните алгоритми и методи за постигане на базово лицево разпознаване. Използвахме ги ефикасно във вградена система и успяхме да запазим сравнително голяма преносимост на устройството.

От тук нататък ние имаме голям избор що се касае до какво искаме да прави устройството. Може да добавим класификатори за най-различни обекти, да приспособим устройството с допълнителни сензори и прочие. Благодарение на факта, че имаме свободен достъп до най-различни обучени класификатори имаме още по-голяма свобода с това каква информация бихме искали да извлечем от един образ и какво да се случва с нея.

Като цяло, този проект е един пример, който демонстрира възможностите на съвременния хардуер в комбинация с ловките алгоритми и оптимизации, които са резултат на редици невероятни проучвания. В общата картина, лицевото разпознаване е само капка в океана на света на машинното обучение. Креативността на един човек е границата що се отнася до разработката на един подобен проект. Вдъхновението за градивните идеи винаги започват от реализацията, че имаш средствата да ги изпълниш.

ДОПЪЛНИТЕЛНИ РЕСУРСИ И ПОЯСНЕНИЯ

1. <https://github.com/thewiffedshot/face_tracking_rpi_camera> - Линк към GitHub репозиторията на проекта.
2. **Raspberry Pi 4** – компютър със способностите на микроконтролер – За повече информация и допълнителна литература по тематиката:  
   <https://en.wikipedia.org/wiki/Microcontroller>  
   <https://en.wikipedia.org/wiki/Raspberry_Pi>
3. **ARM** – вид процесорна архитектура: <https://en.wikipedia.org/wiki/ARM_architecture>
4. **Linux** – популярна операционна система: <https://en.wikipedia.org/wiki/Linux>
5. **Raspberry Pi фондацията**:  
   <https://en.wikipedia.org/wiki/Raspberry_Pi_Foundation>   
   <https://www.raspberrypi.org/>
6. **Python – език за програмиране**:  
   <https://en.wikipedia.org/wiki/Python_(programming_language)>   
   <https://www.python.org/>
7. **Зависимости** – често използвано като синоним на „библиотеки“ или програмни модули, съдържащи определени функционалности, които може да се използват на готово в други програмни модули:  
   <https://en.wikipedia.org/wiki/Coupling_(computer_programming)>
8. **Pip** – Софтуерен пакет, който идва с инсталацията на Python за автоматизирано инсталиране на библиотеки/модули:  
   <https://en.wikipedia.org/wiki/Pip_(package_manager)>
9. **OpenCV** – Библиотека специализираща в машинно обучение и компютърно зрение:  
   <https://opencv.org/>   
   <https://github.com/opencv/opencv>   
   <https://en.wikipedia.org/wiki/OpenCV>
10. **Mашинно обучение** – Науката за компютърни алгоритми, които може да се подобряват автоматично чрез опит и използването на данни:  
    <https://en.wikipedia.org/wiki/Machine_learning>
11. **CAD** програма – Това е софтуер разработен с цел да подпомага инженерно проектиране на детайли и графични модели:  
    <https://en.wikipedia.org/wiki/Computer-aided_design>  
    <https://bg.wikipedia.org/wiki/%D0%9A%D0%BE%D0%BC%D0%BF%D1%8E%D1%82%D1%8A%D1%80%D0%BD%D0%BE_%D0%BF%D1%80%D0%BE%D0%B5%D0%BA%D1%82%D0%B8%D1%80%D0%B0%D0%BD%D0%B5>
12. **Blender** – CAD програма с отворен код и редица възможности:  
    <https://www.blender.org/>
13. **3D принтиране** – Процес на конструиране на триизмерен обект от дигитален 3D модел или CAD модел:  
    <https://en.wikipedia.org/wiki/3D_printing>
14. **CUDA Slicer** – Програма за преобразуване на CAD 3D модел в машинно-специфичен код за 3D принтерни машини:  
    <https://ultimaker.com/software/ultimaker-cura>
15. **Сервомотор (серво)** – Мотор на който може да се задава конкретна позиция (ъгъл), към която да се завърти. Най-често използвани за механично управление:   
    <https://bg.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B2%D0%BE%D0%BC%D0%BE%D1%82%D0%BE%D1%80>  
    <https://en.wikipedia.org/wiki/Servomotor>
16. **Импулсна модулация** – Носещият сигнал се излъчва под формата на поредица от импулси и се променят параметрите на тази поредица (амплитуда, честота, фаза, широчина) с цел шифроване и изпращане на информация. В нашие случай се използва конкретно модулация на импулсната широчина:  
    <https://en.wikipedia.org/wiki/Pulse-width_modulation>   
    <https://bg.wikipedia.org/wiki/%D0%9C%D0%BE%D0%B4%D1%83%D0%BB%D0%B0%D1%86%D0%B8%D1%8F_(%D1%82%D0%B5%D1%85%D0%BD%D0%B8%D0%BA%D0%B0)>
17. **Рискание и тангаж** – Това са основни понятия взети от авиодинамиката. Тези две понятия описват ъгъла на ротация по отделните оси на летателния апарат. Тоест, тези стойности определят неговото насочване в пространството. **Рисканието** (курсът) определя ъгъла на завъртане около оста сочеща вертикално нагоре от центъра на основата на устройството ни. **Тангажът** определя ъгълът на вертикален наклон на устройството ни. Това ще рече ъгъла на ротация около оста ортогонална на вектора сочещ напред от обектива и вектора сочещ по вертикалата на устройството.  
      
      
      
      
      
      
      
      
    *Фиг. 31 – Визуализация на понятията за насока спрямо осите на устройството*  
      
      
    За повече информация:  
    <https://clubaurora.org/%D0%BD%D0%B0%D1%83%D0%BA%D0%B0/%D0%BA%D0%BE%D0%BE%D1%80%D0%B4%D0%B8%D0%BD%D0%B0%D1%82%D0%BD%D0%B8-%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B8-%D0%B8-%D1%81%D0%B8%D0%BB%D0%B8-%D0%B2-%D0%B0%D0%B5%D1%80%D0%BE%D0%B4%D0%B8%D0%BD%D0%B0/>   
    <https://en.wikipedia.org/wiki/Aircraft_principal_axes>
18. **GPIO** – Представлява пин/пинове за вход/изход на интегрална схема или електрическа платка:  
    <https://en.wikipedia.org/wiki/General-purpose_input/output>
19. **Каскаден класификатор**: <https://en.wikipedia.org/wiki/Cascading_classifiers>   
    <https://en.wikipedia.org/wiki/Statistical_classification>
20. **Компютърно зрение**:  
    <https://en.wikipedia.org/wiki/Computer_vision>
21. **Boosting (machine learning)**:  
    <https://en.wikipedia.org/wiki/Boosting_(machine_learning)>
22. **AdaBoost**:  
    <https://en.wikipedia.org/wiki/AdaBoost>  
    <https://machinelearningmastery.com/boosting-and-adaboost-for-machine-learning/>
23. **Weak Learners**:  
    <https://machinelearningmastery.com/strong-learners-vs-weak-learners-for-ensemble-learning/>
24. **Текстура** – Значителен термин в компютърната графика, почти винаги синоним на дигитално изображение:  
    <https://bg.wikipedia.org/wiki/%D0%A2%D0%B5%D0%BA%D1%81%D1%82%D1%83%D1%80%D0%B8%D1%80%D0%B0%D0%BD%D0%B5>   
    <https://en.wikipedia.org/wiki/Texture_mapping>
25. **Билинейна интерполация** – Удължен метод от числовия анализ за линейна интерполация в две измерения:  
    <https://en.wikipedia.org/wiki/Bilinear_interpolation>

ИЗПОЛЗВАНА ЛИТЕРАТУРА И ИЗТОЧНИЦИ

1. *Paul Viola and Michael Jones* Rapid Object Detection using a Boosted Cascade of Simple Features (2001):  
   <https://web.iitd.ac.in/~sumeet/viola-cvpr-01.pdf>
2. *Girija Shankar Behera (2020)* Face Detection with Haar Cascade. *Towards Data Science blogpost*: <https://towardsdatascience.com/face-detection-with-haar-cascade-727f68dafd08>
3. Kelvin Salton do Prado (2017) Face Recognition: Understanding LBPH Algorithm. *Towards Data Science blogpost:* <https://towardsdatascience.com/face-recognition-how-lbph-works-90ec258c3d6b>
4. <https://bg.wikipedia.org/wiki/Raspberry_Pi>
5. <https://en.wikipedia.org/wiki/OpenCV>
6. <https://en.wikipedia.org/wiki/Deadband>
7. *Matti Pietikäinen (2010)* Local Binary Patterns. *Scholarpedia:*<http://www.scholarpedia.org/article/Local_Binary_Patterns>
8. OpenCV docs:  
   <https://docs.opencv.org/master/d6/d00/tutorial_py_root.html>
9. *Luciano Ramalho (O’Reilly, 2014)* Fluent Python: Clear, Concise, and Effective Programming