

**ТЕХНОЛОГИЧНО УЧИЛИЩЕ ЕЛЕКТРОННИ СИСТЕМИ**  
**към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ**

**ДИПЛОМНА РАБОТА**

Тема: Система за анализиране на тенис мач/тренировка.

Дипломант:

Алекс Узунов

Научен ръководител:

гл. ас. Росен Петков

СОФИЯ

2021





# УВОД

Тенисът в днешно време е много динамичен и интензивен спорт. От страна на неговите почитатели, този спорт е много интересен и понякога даже напрегнат за гледане, особено когато в даден мач участва ваш любим състезател.

Но за професионалните състезатели това не е просто игра. За да може да ги видите в тяхната най-добра форма те тренират много усилено и следят всички технически показатели в играта си, за да бъде изчистена в най-голям процент. Използвайки всякаква модерна технология, те успяват да си водят статистика за всеки един собствен удар и да си правят изводи за грешките, които правят най-често. Освен това, по време на мачове, има всякакви технологии, които водят статистика, за играчите, която бива предоставяна на телевизионните зрители, отделно има и Hawk-Eye – система, която се използва в случаите, че играч иска да оспорва съдийско решение и смята, че то е грешно. Тази система, визуализира на базата на запис, който е бил заснет моменти преди случката, дали даденият удар е бил вътре в очертанията на корта или не.

Такива системи не са за свободна употреба и са много скъпи, за самите турнири, за това и не всички турнири от АТР/WТА тура ги използват. Пример за това е Sofia Open 2020, турнир от серията АТР250, който беше в София – Арена Армеец. Поради тежката обстановка и спад в бюджета, турнирните организатори не успяха да платят таксата нужна за използването на Hawk-Eye системата.

Основната цел на това задание е да се направи бюджетна система за анализиране на мач или тренировка, като се използват open-source технологии, проекти и приложения.

# Първа глава

Проучване за изработването на система за  
анализ чрез object detection и системи,  
които вече съществуват.

## **1.1 Технологии за създаване на система за анализ чрез object detection.**

### **1.1.1 Технологии, използвани за object detection**

**1.1.1.1 SSD: Single Shot Detection [1].** Чрез дълбока невронна мрежа, SSD дискретизира изходното пространство на ограничаващите кутии в набор от кутии по подразбиране в различни мащаби и съотношения. Докато прогнозира, невронната мрежа генерира резултати (процент) на присъствието на всеки един инстанциран обект в кутиите по подразбиране и прави корекции върху ограничаващите кутии, за да са по-точни. Също така прави прогнози от множество модели, с различна разпределителна способност, за да обработва по „естествен“ начин обекти с различни размери. Този модел е доста опростен по отношение на методите, които изискват обектни предложения, тъй като тука този модел елиминира напълно генерирането на предложения и последващият етап на преизбиране на пиксели или



характеристики и капсулира всички изчисления в една мрежа. Това е главната причина да е лесен за обучение и интегриране в системи, които изискват object detection.

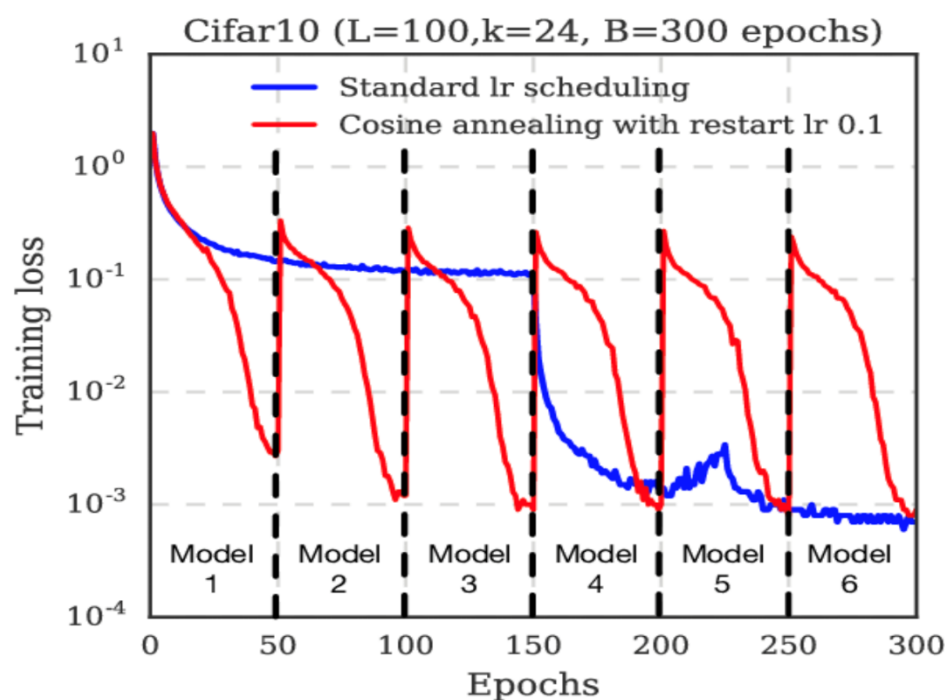
#### 1.1.1.2 YOLO: You Look Only Once.

- **YOLO V1 [2]:** Това е много бърз метод за засичане на обект, тъй като преработва видеа със скорост 45 кадъра в секунда. YOLO е и много прост, като методика. Това е една конволюционна мрежа, която едновременно предсказва множество ограничаващи полета и вероятности за класовете. За разлика от други техники, които използват плъзгащ се екран, за да извлекат информацията, YOLO разглежда цялата картина по време на трениране и тестване, като резултат от което може да енкодира контекстово информацията за класовете, така както и тяхното присъствие. Също така прави и двойно по-малко грешки от Fast R-CNN.
- **YOLO V2 [3]:** Идеята на втората версия е да се намалят грешките при намиране на позиция на обект. Промените в архитектурата, които са направени са следните:
  - Добавя се пакетно нормализиране към архитектурата, което увеличава сходството на модела, който води за

по-бързо обучение. Това подобрява с около 2% точността.

- Класификатор на висока резолюция се добавя, защото в предходната версия, входната информация е с резолюция  $224 \times 224$ , но при самата детекция се удвоява. Това е водело до намаляване на процента на точност. В новата версия тренираният модел е с по-висока резолюция ( $448 \times 448$ ). По този начин процента се повишава с около 4%.
- Докато старата версия използва напълно свързани слоеве, за да предскаже ограничаващите полета, вместо да предсказва координатите директно от конволюционната мрежа, подобно на Fast R-CNN или Faster R-CNN, новата версия използва така наречените „кутии-котва“, за да предскаже ограничаващите кутии.
- **YOLO V3 [4]:** Подновената версия използва вариация на Darknet, която първоначално използва 53-слова мрежа, обучена на Imagenet. За разпознаването на обекти върху мрежата има още 53 слоя, което ни дава 106 слоя изцяло извита основна архитектура, която е по-добра от предишната. Но това е и причината YOLO V3 да е по-бавна от V2.

- **YOLO V4 [5]:** YOLO V4 е едноетапен модел за откриване на обекти, който подобрява YOLO V3 с няколко трика и модула. Те включват т. нар. „косинусно отгряване“ като график за скоростта на обучение (фигура 1.1), CSPDarknet53, който играе ролята на „backbone“ на YOLO V4 и е отделна конволюционна невронна мрежа.



Фигура 1.1 Сравнение между нормален график за скоростта на обучение и график, базиран на концепцията за косинусно отгряване.

- **YOLO V5 [6]:** Гръбначният модел се използва главно за извличане на важни характеристики от дадено изображение.

## **1.2 Програмни езици, използвани за AI и математически анализ.**

**1.2.1 Python [7]:** „Python“ е интерпретируем, интерактивен, обектно-ориентиран език за програмиране. Предлага добра и лесна структура за използване и разработване на големи приложения. Също така притежава вградени сложни типове данни като гъвкави масиви и речници, за които би отнело дни да бъдат написани ефективно на „C“. В днешно време „Python“ е един от водещите езици използвани за машинно самообучение. Една от главните библиотеки свързани със самообучение в езика е „TensorFlow“.

**1.2.2 Haskell [8]:** Haskell (Хàскъл) е функционален език за програмиране. В частност, той е полиморфично статично-типизиран, „мързелив“ (нищо не се прави, докато не се наложи), чисто функционален език, доста различен от повечето езици за програмиране. Функциите в Haskell нямат странични ефекти. Съществува ясна конструкция за представяне на странични ефекти, независима от типа на функциите. Една чиста функция може да върне страничен ефект, който впоследствие се изпълнява, подобно на нечистите (impure) функции при другите езици. Haskell има силна, статична система от типове, базирана на системата от типове на Хиндли–Милнър. Поради близостта си

с математиката и богатството от математически библиотеки езикът е широко използван за решаване на кратки математически задачи.

**1.2.3 LISP [9]:** LISP (LISt Processing language - "език за обработка на списъци") представлява съвкупност от програмни езици и данни. Въз основа на него са произлезли много диалекти, най-известните от които са Scheme и Common Lisp. LISP е вторият програмен език от високо ниво в историята след FORTRAN. Като един от най-ранните езици за програмиране, LISP внедрява много идеи в компютърните науки, включително дървовидни структури от данни, автоматично управление на съхранението, динамично въвеждане, условни функции, функции от по-висок ред, рекурсия, самохостиращ компилатор и read-eval-print цикъл. LISP е едно от основните програмни средства за моделиране на различни аспекти в изкуствения интелект.

**1.2.4 R [10]:** R е програмен език и развойна среда за статистическа и математическа обработка на данни. Езикът включва възможности за работа с различни типове данни – числови, низови и логически (булеви), както и с обектни структури като: вектори, фактори, списъци, матрици и датафреймове, if-else, for, repeat и други. R и библиотеките му имплементират голяма разновидност от статистически и графични технологии, включително линейни

и нелинейни модели, класически статистически тестове, времеви анализи, класификации и други. Широко се използва в изкуствения интелект от нов стил, включващ статистически изчисления, числени анализи, използването на Байесовски извод, невронни мрежи и като цяло машинно обучение.

**1.2.5 Julia [11]:** Julia е високоефективен, динамичен език за програмиране с общо предназначение, като същевременно първоначално е проектиран за цифрови и технически изчисления. Използва многократно изпращане като парадигма, което улеснява изразяването на много обектно-ориентирани и функционални модели на програмиране. Пакетът MLJ.jl предоставя унифициран интерфейс към обичайните алгоритми за машинно обучение, които включват обобщени линейни модели, дървета на решенията и клъстериране. Flux.jl и Knet.jl са мощни пакети за дълбоко обучение. Пакети като Metalhead, ObjectDetector и TextAnalysis.jl предоставят готови за използване предварително обучени модели за общи задачи. AlphaZero.jl осигурява висока ефективност на изпълнението на алгоритмите за обучение с утвърждение от AlphaZero, а Turing.jl е пакет за вероятностно програмиране.

### **1.3 Подходящи развойни среди за машинно обучение.**

**1.3.1 Spyder [12]:** Spyder е cross-platform open-source развойна среда, която се използва за анализиране на данни. Плюсовете на тази среда са, че може да се визуализират графики, също както в MATLAB. Spyder много наподобява PyCharm, но е с различен UI.

**1.3.2 Visual Studio Code [13]:** Visual Studio Code е редактор на програмен код за Windows, Linux и OS X. Това е първият редактор на Microsoft, който може да се ползва под Linux и macOS. Той също така дава възможност за персонализиране, което означава, че потребителите могат да променят темата на редактора, клавишните комбинации, настройките и други. Редакторът е продукт на Microsoft и е безплатен, публично достъпен за преглед. Базиран е на Electron, който е базиран на Chromium, използван да разгръща io.js приложения за десктопа.

**1.3.3 PyCharm [14]:** PyCharm е развойна среда, която се използва за програмиране на Python. Разработен е от чешката фирма JetBrains. Предоставя анализ на код, графичен дебъгер, възможност за правене на unit тестове, интеграция със системи за контрол на версиите, отлична поддръжка на Anaconda.

#### **1.4 Съществуващи разработки на системи за следене на обекти в тениса.**

**1.4.1 TrackNet [15]:** TrackNet представлява дълбока невронна мрежа за проследяване на тенис топка от излъчвани видеоклипове, в които

образите на топката са малки, размазани и понякога дори блокирани. Тя е базирана на т.нар. топлинна карта (heatmap) и е обучена не само да разпознава образа на топката от един кадър, но и да научава схеми на летене от последователни кадри. TrackNet прави изображения с размер 640x360, за да генерира топлинна карта за откриване или от един кадър, или от няколко последователни кадъра, за да позиционира топката. Мрежата се оценява на видеоклипа на финала на индивидуалните мачове за мъже на Лятната Универсиада в Тайпе през 2017 година, който е достъпен в YouTube. Точността, чувствителността и F-мярката на TrackNet достигат съответно 99,7%, 97,3% и 98,5%. За да се предотврати свръх подготовка, още 9 видеоклипа са частично класифицирани заедно с подмножество от предишен набор от данни, за да се приложи десетократно кръстосано валидиране, а точността, чувствителността и F-мярката са съответно 95,3%, 75,7% и 84,3%. Освен това е приложен и конвенционален алгоритъм за обработка на изображения за сравнение с TrackNet.

**1.4.2 Hawk-Eye [16]:** Hawk-Eye е система за компютърно зрение, използвана в много спортове като крикет, тенис, галски футбол, бадминтон, хвърляне, ръгби, футбол и волейбол, за визуално проследяване на траекторията на топката и показване на профил на нейния статистически най-вероятен път. Системата работи



чрез шест (понякога седем) високоефективни камери, обикновено разположени от долната страна на покрива на стадион, които проследяват топката от различни ъгли. След това видеото от шестте камери се триангулира и комбинира, за да се създаде триизмерно представяне на траекторията на топката. За тенис има десет камери. Системата бързо обработва видео емисиите от камерите и проследява на топката. Съществува хранилище за данни, което съдържа предварително дефиниран модел на игралната зона и включва данни за правилата на играта. Системата генерира графично изображение на пътя на топката и зоната за игра, което означава, че информация може да бъде предоставена на съдии, телевизионни зрители или треньорски персонал в почти реално време. Системата за проследяване е комбинирана с back-end база данни и възможности за архивиране, така че да е възможно да се извлекат и анализират тенденции и статистика за отделни играчи, игри и т.н. Hawk-Eye се отличава с точност до 3,6 милиметра и в повечето случаи на него се разчита като безпристрастно второ мнение в спорта.

# Втора глава

Изисквания към система за анализиране на  
тенис мач/тренировка. Избор на език за  
програмиране и софтуерни средства.

## **2.1 Функционални изисквания към система за анализиране на тенис мач/тренировка:**

**2.1.1** Системата трябва да може да предоставя възможност за заснемане на тенис мач или тренировка и да разпознава различните обекти (тенис топка, която е в разиграване; тенис ракетите на играчите и самите тях). Чрез разпознатите обекти да се прави статистика спрямо различни ситуации и да е насочена към конкретни особености: Видове удари, „гореща точка“ в различни ситуации, средна височина на разиграваната топка в различни ситуации, най-използван удар в различни ситуации.

**2.1.2** Чрез приложението на системата да може да се визуализира направената статистика за средната височина на тенис топката в различни аспекти на играта, да може да визуализира местата, на които попада тенис топката върху тенис корта.

**2.1.3** Чрез приложението на системата да се определя от потребителя, кога приключило разиграване, гейм, сет или мач, за да може да се отбелязват данните в съответните места.

## **2.2 Избрани програмни средства и развойна среда съобразени с проекта:**

### **2.2.1 Visual Studio Code:**

**2.2.1.1 Поддържан от macOS, Linux, и Windows:** Visual Studio Code поддържа macOS, Linux и Windows, като с това дава

възможност на потребителя да го използва, без значение на коя от трите платформи се намира.

**2.2.1.2 Поддържа множество програмни езици:** Visual Studio Code поддържа над 30 програмни езика - Batch, C++, C#, Clojure, CoffeeScript, CSS, DockerFile, F#, Go, HTML, Jade, Java, JavaScript, JSON, HandleBars, Ini, Lua, Less, Makefile, Markdown, Objective-C, Perl, PHP, PowerShell, Python, R, Razor, Ruby, Rust, SQL, Sass, TypeScript, Visual Basic, XML.

**2.2.1.3 Отворен и безплатен за използване:** Visual Studio Code е предимно с отворен код и това е безпрецедентно предимство. Преминаването в отворен код е чудесен избор с цел да се увеличи ангажираността на общността. Въпреки че не всички потребители на Visual Studio Code допринасят за неговата кодова база, те получават определено усещане за единство.

**2.2.1.4 Прост за използване:** От първите стъпки до инсталирането на нови разширения, всичко във Visual Studio Code е просто и интуитивно. Благодарение на разширяемата си архитектура, Visual Studio Code, дори когато е само редактор на код, може да послужи като ценна алтернатива на други по-сложни IDE-та.

**2.2.1.5 Здрава и гъвкава архитектура:** В архитектурно отношение Visual Studio Code съчетава най-доброто от уеб и специфични технологии (от гледна точка на езика и/или конкретната операционна система). Използвайки Electron, Visual Studio Code съчетава уеб технологии като JavaScript и Node.js със скоростта и гъвкавостта на родните приложения. Visual Studio Code използва по-нова, по-бърза версия на същия HTML-базиран редактор с индустриална мощност, който е задвижвал облачния редактор „Monaco“, „F12 Tools“ на Internet Explorer и други проекти. Освен това Visual Studio Code използва архитектура за услуги на инструменти, която му позволява да се интегрира с много от същите технологии, които захранват Visual Studio, включително Roslyn за .NET, TypeScript, механизма за отстраняване на грешки на Visual Studio и други.

**2.2.1.6 Дава възможност за настройване на всяка особеност по вкуса на потребителя:** Visual Studio Code позволява на потребителят да персонализира всяка налична функция по негов вкус и да инсталира произволен брой разширения. Освен това, той включва публичен модел на

разширяемост, който позволява на разработчиците да създават и използват разширения и богато да персонализират своето преживяване при редактиране, изграждане и отстраняване на грешки.

**2.2.2 Python:** Python притежава множество вградени библиотеки, от които много са свързани с изкуствен интелект и машинно обучение. Някои от тях са TensorFlow (което е библиотека от невронни мрежи на високо ниво), scikit-learn (за извличане на данни, анализ на данни и машинно обучение), pylearn2 (по-гъвкав от scikit-learn) и др. Това са само част от библиотеките, които могат да бъдат използвани. Освен това Python има лесна реализация за OpenCV.

- **Четим и поддържаме код.**

Докато разработва софтуерно приложение, разработчикът трябва да се съсредоточи върху качеството на неговия изходен код, за да опрости поддръжката и актуализациите. Правилата на синтаксиса в Python позволяват да се изразяват концепции, без да се пише допълнителен код. В същото време Python, за разлика от другите езици за програмиране, набляга на четливостта на кода и ви позволява да използвате английски ключови думи вместо пунктуации. Следователно Python може да се използва за създаване на персонализирани приложения, без да се пише допълнителен код. Четливата и чиста кодова

база помага да се поддържа и актуализира софтуера, без да се влагат допълнително време и усилия.

- **Множество парадигми на програмиране.**

Подобно на други съвременни езици за програмиране, Python също поддържа няколко парадигми на програмиране. Той поддържа изцяло обектно-ориентирано и структурирано програмиране. Също така неговите езикови функции поддържат различни концепции във функционалното и аспектино-ориентираното програмиране. В същото време Python разполага и с динамична система от типове и автоматично управление на паметта. Парадигмите на програмиране и езиковите функции помагат в това Python да се използва за разработване на големи и сложни софтуерни приложения.

- **Съвместим с основните платформи и системи.**

Понастоящем Python поддържа много операционни системи. Може дори да се използват интерпретатори на Python, за да се стартира кода на определени платформи и инструменти. Освен това Python е интерпретируем език за програмиране. Това позволява да се стартира един и същ код на множество платформи без повторна компилация. Следователно не е необходимо кодът да се прекомпилира, след като е направена каквато и да е промяна. Модифицираният код на приложението

може да се стартира без прекомпилиране и незабавно да се провери въздействието на промените, направени в кода. Тази функция улеснява правенето на промени в кода, без да се увеличава времето за разработка.

- **Здрава стандартна библиотека.**

Неговата голяма и здрава стандартна библиотека прави Python по-резултатен спрямо други езици за програмиране. Стандартната библиотека позволява да се избира от широка гама модули според точните нужди на разработчика. Освен това всеки модул позволява да се добави функционалност към Python приложението, без да се пише допълнителен код. Например, докато се разработва уеб приложение в Python, може да се използват специфични модули за внедряване на уеб услуги, извършване на низови операции, управление на интерфейса на операционната система или работа с интернет протоколи. Може дори да се събира информация за различни модули.

- **Много софтуерни рамки и инструменти с отворен код.**

Като език за програмиране с отворен код, Python помага да се намалят значително разходите за разработка на софтуер. Може дори да се използват няколко Python рамки, библиотеки и инструменти за разработка с отворен код, за да се намали



времето за разработка, без да се увеличават разходите за разработка. Освен това има възможност за избор от широка гама Python рамки и инструменти за разработка с отворен код според точните нужди на разработчика. Например, може да се опрости и ускори разработването на уеб приложения, като използвате стабилни уеб рамки на Python като Django, Flask, Pyramid, Bottle и CherryPy. По същия начин може да се ускори разработката на графичен потребителски интерфейс (GUI) за настолни компютри, използвайки Python GUI рамки и набори от инструменти като PyQt, PyJs, PyGUI, Kivy, PyGTK и WxPython.

- **Опростена е сложната разработка на софтуер.**

Python е език за програмиране с общо предназначение. Следователно може да се използва за разработване както на настолни, така и на уеб приложения. Също така може да се използва за разработване на сложни научни и цифрови приложения. Python е проектиран с функции за улесняване на анализа на данните и визуализацията. Разработчикът може да се възползва от функциите за анализ на данни на Python, за да създаде персонализирани решения за големи данни, без да влага допълнително време и усилия. В същото време библиотеките за визуализация на данни и

приложно-програмен интерфейс (API), предоставени от Python, помагат да се визуализират и представят данни по по-привлекателен и ефективен начин. Много разработчици на Python дори го използват, за да изпълняват задачи, свързани с изкуствен интелект (AI) и обработка на естествен език.

- **Разработка чрез тестове.**

Python може да се използва за бързо създаване на прототип на софтуерно приложение. Също така софтуерното приложение може да се изгради директно от прототипа, просто като се рефакторира кода на Python. Python дори улеснява извършването на кодиране и тестване едновременно, като възприема подхода за разработка чрез тестове (TDD). Може лесно да се напишат необходимите тестове, преди да се напише код, и тези тестове да се използват за непрекъсната оценка на кода на приложението. Тестовите могат да се използват и за проверка на това дали приложението отговаря на предварително дефинирани изисквания въз основа на изходния код.

**2.2.2.1 OpenCV [17]:** OpenCV е библиотека, състояща се от програмни функции, които са насочени главно към компютърно зрение в реално време. Плюс е, че е безплатна за употреба и също така, че предоставя ускорение на графичният процесор по време на

изпълнение на операции в реално време. OpenCV за Python не е нищо друго освен клас на обвивка за оригиналната библиотека на C++, която да се използва с Python. Използвайки това, всички масиви на OpenCV се преобразуват в масиви на NumPy. Това улеснява и интеграцията му с други библиотеки, които използват NumPy, като например Matplotlib или SciPy. Също така Python е по-бавен от C++, но пък е по-лесен за разработка. Точно тези класове обвивки, които са споменати по-горе, спомагат за по-бързата преработка на кода, тъй като реално са C++ базирани, но под формата на Python script.

- **Поддръжка на няколко езика.**

OpenCV поддържа редица езици за програмиране, включително Android SDK, Java, Python, C++ и MATLAB, а освен това има C++, Python, Java и MATLAB интерфейси. Библиотеката е написана на оптимизиран C и има способността да се възползва от многоядрените процесори.

- **Множество алгоритми.**

OpenCV има над 2500 оптимизирани алгоритми и включва изчерпателен набор както от класически, така и от модерни алгоритми за компютърно зрение и машинно обучение. Библиотеката е и пълна универсална библиотека за машинно обучение (MLL), която се фокусира върху статистическо разпознаване на модели, а също така и клъстериране, и е

изключително полезна за задачите на компютърното зрение. Тази подбиблиотека обаче е достатъчно обща, за да се използва за всеки проблем, свързан с машинното обучение.

- **Разнообразие от проекти.**

Алгоритмите в тази библиотека могат да се използват за различни видове проекти, като откриване и разпознаване на лица, идентифициране на обекти, класифициране на човешки действия във видеоклипове, проследяване на движенията на камерата и определяне на движещи се обекти. В допълнение, той също се използва за извличане на 3D модели на обекти, създаване на 3D облаци от точки от стерео камери, съчетаване на изображения, за да се получи изображение с висока разделителна способност, намиране на подобни изображения от база данни с изображения, премахване на червени очи от изображения, направени с помощта на светкавица, проследяване на движенията на очите, разпознаване на пейзажи и установяване на маркери, които да го припокриват с обогатена реалност и много други.

#### **2.2.2.2 NumPy [18]:**

NumPy е една от най-популярните библиотеки на Python за работа с многомерни масиви и матрици, съдържаща колекция от

математически функции от високо ниво. Полезен е за фундаментални научни изследвания в машинното обучение.

- **По-малко памет за съхранение на данни.**

NumPy масивите отнемат значително по-малко количество памет в сравнение със списъците на Python. Библиотеката също така предоставя механизъм за определяне на типовете данни на съдържанието, което позволява по-нататъшна оптимизация на кода.

- **Бърза производителност.**
- **Удобен за работа.**

**2.2.2.3 Matplotlib [19]:** Matplotlib е библиотека за графика за Python и NumPy. Представява обектно-ориентиран интерфейс за приложно програмиране за вграждане на графики в приложения, използващи набори от графичен потребителски интерфейс с общо предназначение като Tkinter, wxPython, Qt или GTK+. SciPy използва Matplotlib.

- **Исключително адаптивна и мощна.**
- **Предоставя прецизен контрол върху графиките.**
- **Голямо разнообразие от методи и функции за генериране на различни видове графики.**
- **Проста и лесна за разбиране за начинаещи.**

- **Осигурява висококачествени изображения и графики в различни формати като png, pdf, pgf и други.**

**2.2.2.4 PyTorch [20]:** PyTorch обвива същия “C” back-end в интерфейс на Python. Но това е нещо повече от обвивка. Разработчиците са го изградили от нулата, за да улеснят писането на модели за програмисти на Python. Основният код на C и C++ на ниско ниво е оптимизиран за стартиране на Python код. Поради тази тясна интеграция се получават:

- **По-добра памет и оптимизация.**
- **По-разумни съобщения за грешка.**
- **По-фин контрол на структурата на модела.**
- **По-прозрачно поведение на модела.**
- **По-добра съвместимост с NumPy.**

Това означава, че може да се пишат силно персонализирани компоненти на невронната мрежа директно в Python, без да се налага да се използват много функции на ниско ниво. Преобразуването на „NumPy“ обекти в „tensors” е включено в основните структури от данни на PyTorch. Това означава, че може лесно да се превключва между обектите на torch.Tensor и обектите numpy.array.

**2.2.3 Electron [21]:** Electron (известен преди като Atom Shell) е софтуерен фреймуърк с отворен код, разработен и поддържан от GitHub. Той позволява разработването на настолни GUI приложения, използващи уеб технологии. Electron е основната рамка на GUI зад няколко проекта с отворен код (в това число Atom, GitHub Desktop, Light Table, Visual Studio Code, Evernote и WordPress Desktop).

**2.2.3.1 Съвместим с macOS, Windows и Linux:** Приложения, разработени чрез Electron, работят на тези три платформи.

**2.2.3.2 Проект с отворен код:** Electron е проект с отворен код, поддържан от GitHub и активна общност от сътрудници.

**2.2.3.3 Сигурност на данните:** Не е нужно да се мисли много, когато се мигрира съществуващо приложение към Electron, тъй като приложението, което се създава, е настолно приложение и данните остават локално в системата. Поради това е възможно да се осигури сигурност на данните. В случай, че трябва да се съхраняват данни в облака, предварително трябва да се провери дали наличната облачна мрежа има достатъчно функции за сигурност, за да се избегнат нежелани изненади.

**2.2.3.4 Достъпност на ниско ниво:** Преди да се започне, трябва също така да се провери дали всички функции, които се предоставят за съответното настолно приложение, също са налични в Electron.

Electron осигурява достатъчен контрол, за да има разширени интерактивни функции във приложението като клавишни комбинации. Той също така осигурява достъпност на ниско ниво до хардуера и компонентите на операционната система.

**2.2.3.5 Достъпност на хардуер:** Разработчиците могат да получат пълен достъп до всички приложно-програмни интерфейси (API) за достъп на ниво хардуер чрез JavaScript/Plugin. Не е необходимо да се правят компромиси за тази функционалност, ако разработчикът има намерение да мигрира към Electron.

**2.2.3.6 Производителност:** Electron процъфтява в този аспект. Ако се полагат подходящи грижи при разработването (да се зарежда само това, от което има нужда), Electron може да покаже някои големи печалби по отношение на производителността в сравнение с native приложения. Electron спестява много време и предоставя повече опции за развитие, като разполага с една кодова база за всички основни платформи.

**2.2.3.7 Управление на кода и приложението:** На собственик на продукт не му е необходимо да поддържа различни екипи за всяка платформа и съответно няма да му се налага да преосмисля своите изисквания с различни екипи. Това също така ще намали одиторската работа, за да е сигурно, че продуктът има една и съща функционалност на различните платформи. На разработчик не му е



нужно да се притеснява за различни кодови бази. Ако срещне грешка на която и да е платформа, може да я поправи в основата на кода. Грешката никога няма да се появи на други платформи. Въпреки това е хубаво да се следят функционалностите на ниво операционна система.

**2.2.3.8 Многократна употреба:** Понеже се използва една кодова база, това означава, че тя може да се използва както за уеб приложения, така и за настолни приложения. Освен това по някакъв начин се използва повторно базовия код на различни платформи, тъй като всичко се разработва веднъж, а се разпространява навсякъде.

**2.2.3.9 Разпространяване:** Това е един от интересните аспекти на Electron. Наличен е модул за пакетизиране, който помага да се обедини цялата кодова база в съответните пакети.

**2.2.3.10 Потребителски интерфейс (UI) и потребителско изживяване (UX):** С уеб технологии (Chromium, Node.js, HTML, CSS и JavaScript), разработчиците са отворени за множество технологии, които осигуряват страхотни потребителски интерфейс (UI) и потребителско изживяване (UX) на всички потребители с голям комфорт. Също така е сигурно, че се предоставя еднакво изживяване на всички потребители на различни платформи.

**2.2.3.11 Разходи и време:** Спестяват се много време и пари за разработка, защото за технологичния стек, който се използва, има

много разработчици, които могат да го направят за по-малко и да постигнат добри резултати. Може да се спести много време, като се използва единичната кодова база и всеки разработчик може да работи върху всичко.

# Трета глава

Програмна реализация на система за  
анализиране на тенис мач/тренировка.

### 3.1 Установяване и обработка на информация

**3.1.1 court.py:** По своето предназначение court.py отговаря за разпознаването на измеренията на тенис корта, съчетавайки съвкупност от функционалности, които OpenCV библиотеката предоставя. Първоначално, при заснемане на съответен кадър, полученото изображение се взима и се конвертира в Grayscale изображение (Фигура 3.1).

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

Фигура 3.1 Конвертиране на изображение в Grayscale изображение чрез  
OpenCV

На полученото изображение като следваща стъпка се прилага така нареченото размиване на Гаус (Gaussian Blur или Gaussian Smoothing), чието предназначение е да се отърве от наличните смущения в изображението - по-специално в тази част от тенис корта, която е закрита от мрежата, която го разделя на две половини (Фигура 3.2).

```
kernel_size = 5  
blur_gray = cv2.GaussianBlur(gray, (kernel_size, kernel_size), 0)
```

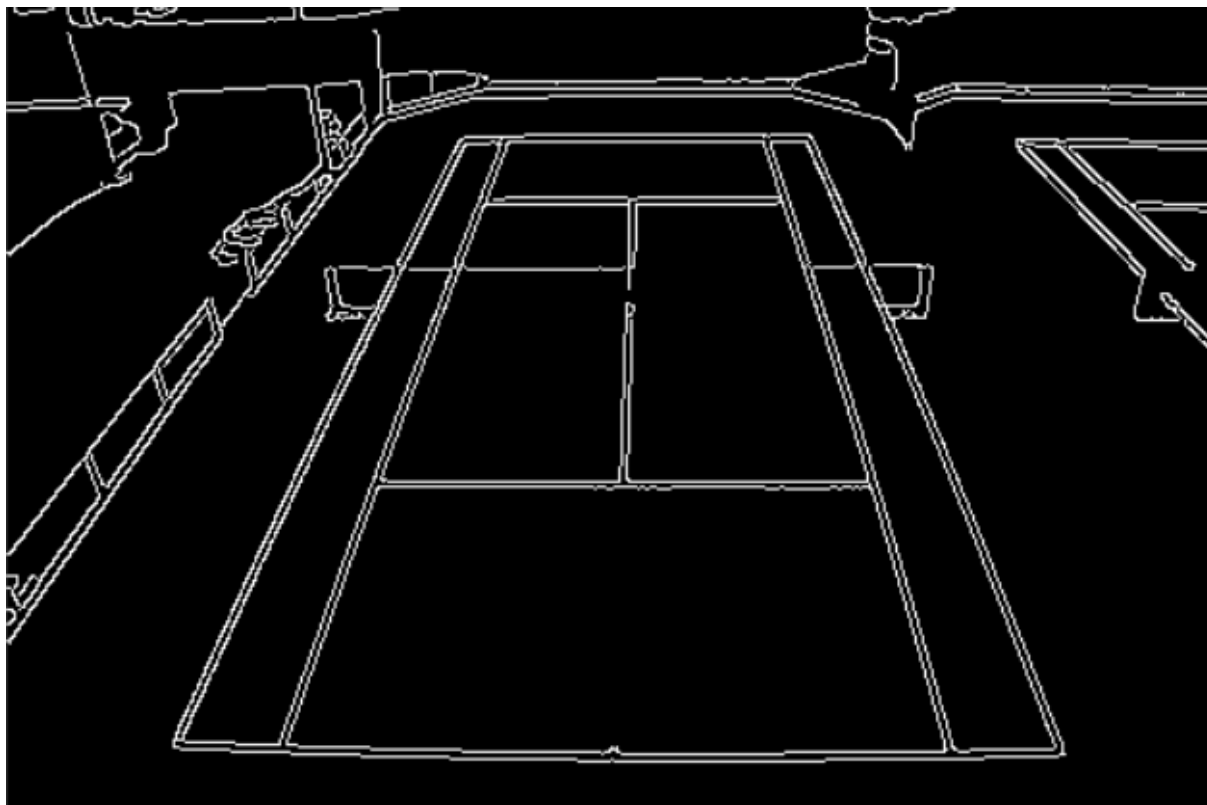
Фигура 3.2 Прилагане на размиване на Гаус върху изображение чрез  
OpenCV

След като изображението е изчистено му се прилага така наречения Детектор на граници на Кани (Canny Edge Detector), който сам по себе си се използва за разпознаване на наличните ръбове в дадено изображение

(Фигура 3.3). Изображението на Фигура 3.4 е приложено с цел да се демонстрира работата на това пособие.

```
edges = cv2.Canny(blur_gray, 50, 250)
```

Фигура 3.3 Прилагане на Canny Edge Detector върху изображение чрез OpenCV



Фигура 3.4 Резултат от прилагането на Canny Edge Detector върху изображение на тенис корт

След получаването на подобен резултат се прилага така наречената Вероятностна трансформация на линии на Хъф (Probabilistic Hough Line Transform), която се използва за разпознаване на прави линии и следователно помага да се открият очертанията на тенис корта. Като краен

резултат функцията връща масив от масиви, съдържащи четири числа, които представляват съответно координати на двете крайни точки на установена линия. С тези координати в наличност съответните линии могат да се разчертаят върху оригиналното изображение (Фигура 3.5).

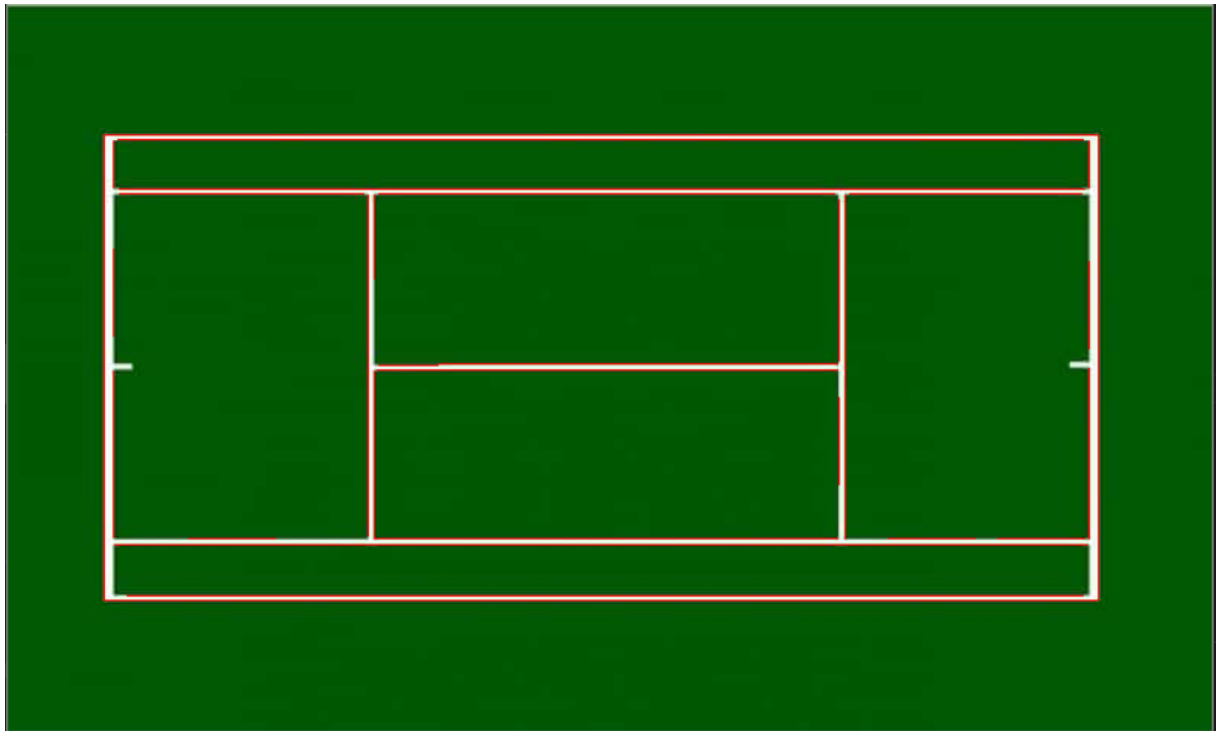
```
lines = cv2.HoughLinesP(edges, 1, np.pi / 180, 30, np.array([]), 20)
print(lines)

for line in lines:
    x1, y1, x2, y2 = line[0]
    cv2.line(img, (x1, y1), (x2, y2), (0, 0, 255), 1)
```

Фигура 3.5 Прилагане на Probabilistic Hough Line Transform върху изображение, преминало през Canny Edge Detector, извеждане на получените като резултат масиви от координати (Фигура 3.6) и разчертаване на съответните линии върху оригиналното изображение


```
[[700 118 700 88]]
[[538 254 538 310]]
[[237 177 237 123]]
[[641 345 697 345]]
[[240 233 278 233]]
[[240 122 535 122]]]
```


Фигура 3.6 Формат на връщаните масиви от координати ( $[x_1, y_1, x_2, y_2]$ )



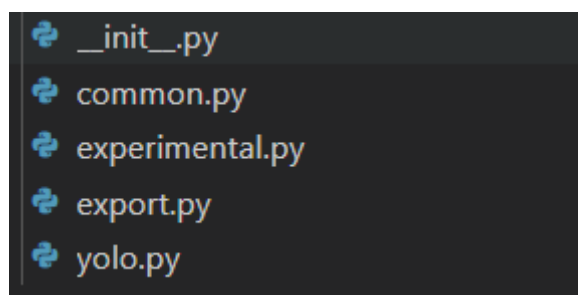
Фигура 3.7 Резултат на промените върху оригиналното изображение след прилагане на кода от Фигура 3.5

**3.1.2 detect.py:** Освен `court.py` в основата на средствата за установяване и обработка на информация стои `detect.py`, който отговаря за извършването на разпознаване на обекти (object detection) в рамките на едно изображение или видео материал (съвкупност от кадри). От своя страна `detect.py` се възползва от други функционалности, разпределени в рамките на проекта в специфични директории.

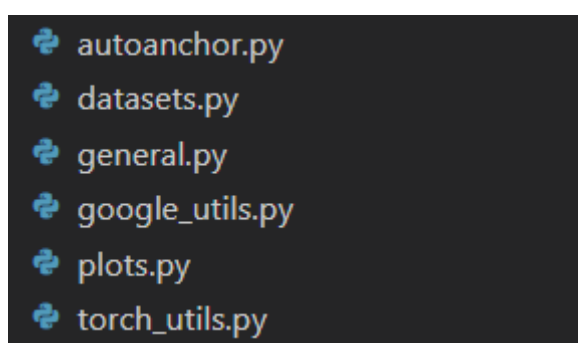
 `models`

 `utils`

Фигура 3.8 Директориите models и utils, от които detect.py черпи конкретни функционалности



Фигура 3.9 Файлове, съдържащи се в models директорията



Фигура 3.10 Файлове, съдържащи се в utils директорията

```
from models.experimental import attempt_load
from utils.datasets import LoadStreams, LoadImages
from utils.general import check_img_size, non_max_suppression, apply_classifier, scale_coords, \
    xyxy2xywh, strip_optimizer, set_logging, increment_path
from utils.plots import plot_one_box
from utils.torch_utils import select_device, load_classifier, time_synchronized
```

Фигура 3.11 Вмъкване на функционалности, използвани в рамките на detect.py

Една от тях е функцията `attempt_load`, намираща се в рамките на файла `experimental.py` на директорията `models`, която отговаря за зареждането на даден модел на невронна мрежа.



```
model = attempt_load(weights, map_location=device)
```

Фигура 3.12 Употреба на attempt\_load в рамките на detect.py

Освен това се използват инстанции на класовете LoadStreams и LoadImages, намиращи се в рамките на файла datasets.py. Инстанции на тези два класа са конфигурирани в рамките на detect.py с цел зареждане на снимков или видео материал (с възможност за няколко камери при заснемане) за обработка от съответния модел.

```
if webcam:
    view_img = True
    cudnn.benchmark = True # set True to speed up constant image size inference
    dataset = LoadStreams(source, img_size=imgsz, stride=stride)
else:
    save_img = True
    dataset = LoadImages(source, img_size=imgsz, stride=stride)
```

Фигура 3.13 Употреба на инстанции на класовете LoadStreams и LoadImages в рамките на detect.py

Функции от файлът general.py в директорията utils намират широко приложение в рамките на detect.py. Функцията check\_img\_size е допълнение при зареждането на даден модел и служи за проверка на размера на съответно изображение (Фигура 3.14). Функцията non\_max\_suppression от своя страна имплементира алгоритъм, който избира един обект от много припокриващи се обекти на базата на определена вероятностна граница (Фигура 3.15). Функцията apply\_classifier се използва за прилагане на даден класификатор с цел предсказване на

класовете на дадени обекти (Фигура 3.16). Функцията `scale_coords` от своя страна се използва за преоразмеряване на координатите, чрез които се определя разположението на обектите в дадено изображение или кадър (Фигура 3.17). Функцията `хуху2хуwh` се използва за конвертиране на формата на определени координати. Тук става въпрос за преминаване от формат `хуху` към `хуwh`, като при `хуху` имаме разположението на най-горната лява точка на обекта и най-долната дясна точка на обекта, а при `хуwh` имаме разположението на най-горната лява точка на обекта, но вместо най-долната дясна точка на обекта имаме неговата широчина и височина (Фигура 3.18). Функцията `increment_path` се използва като разпределителна мярка при разпознаването на обекти, за да може резултатите от различни изпълнения на програмата да се съхраняват на различни места (Фигура 3.19).

```
imgsz = check_img_size(imgsz, s=stride) # check img_size
```

Фигура 3.14 Употреба на `check_img_size` в рамките на `detect.py`

```
# Apply NMS
pred = non_max_suppression(pred, opt.conf_thres, opt.iou_thres, classes=opt.classes, agnostic=opt.agnostic_nms)
t2 = time_synchronized()
```

Фигура 3.15 Употреба на `non_max_suppression` в рамките на `detect.py`

```
if classify:
    pred = apply_classifier(pred, modelc, img, im0s)
```

Фигура 3.16 Употреба на `apply_classifier` в рамките на `detect.py`

```
# Rescale boxes from img_size to im0 size
det[:, :4] = scale_coords(img.shape[2:], det[:, :4], im0.shape).round()
```

Фигура 3.17 Употреба на scale\_coords в рамките на detect.py

```
xywh = (xyxy2xywh(torch.tensor(xyxy).view(1, 4)) / gn.view(-1).tolist()) # normalized xywh
```

Фигура 3.18 Употреба на xyxy2xywh в рамките на detect.py

```
# Directories
save_dir = Path(increment_path(Path(opt.project) / opt.name, exist_ok=opt.exist_ok)) # increment run
(save_dir / 'labels' if save_txt else save_dir).mkdir(parents=True, exist_ok=True) # make dir
```

Фигура 3.19 Употреба на increment\_path в рамките на detect.py

От друга страна, detect.py се разпорежда само с една функция от plots.py на директорията utils. Това е функцията plot\_one\_box, която се използва за разчертаване на квадрати около разпознатите обекти (Фигура 3.20).

```
plot_one_box(xyxy, im0, label=label, color=colors[int(cls)], line_thickness=3)
```

Фигура 3.20 Употреба на plot\_one\_box в рамките на detect.py

Използват се още 3 функции, които detect.py заимства от torch\_utils.py на директорията utils. Функцията select\_device отговаря за избор на устройството, на което ще се изпълнява разпознаването на обекти (Фигура 3.21). Функцията load\_classifier като допълнение на apply\_classifier се използва за зареждане на даден класификатор (Фигура 3.22). Функцията time\_synchronized от своя страна играе ролята на таймер (Фигура 3.23).

```
device = select_device(opt.device)
```

Фигура 3.21 Употреба на select\_device в рамките на detect.py

```
modelc = load_classifier(name='resnet101', n=2) # initialize
```

Фигура 3.22 Употреба на load\_classifier в рамките на detect.py

```
# Inference
t1 = time_synchronized()
pred = model(img, augment=opt.augment)[0]

# Apply NMS
pred = non_max_suppression(pred, opt.conf_thres, opt.iou_thres, classes=opt.classes, agnostic=opt.agnostic_nms)
t2 = time_synchronized()
```

Фигура 3.23 Употреба на time\_synchronized в рамките на detect.py

Моделът, който се използва за разпознаване на обекти, е трениран на така наречения COCO (Common Objects In Context) набор от данни (dataset) на Microsoft. Видът на обектите, които може да разпознава, се означава и съответно се определя с помощта на определени числа, които могат да се заимстват с помощта на файл, наречен coco.names. С оглед на това detect.py има достъп до тези числа при установяване на разпознатите обекти и това спомага в имплементацията на функционалности, свързани с разпознаване на топка за тенис (Фигура 3.24).

```
if (int(cls) == 32):
    width = abs((int(xyxy[0]) + int(xyxy[2])))
    height = abs((int(xyxy[1]) + int(xyxy[3])))
    centers.append((round(width / 2), round(height / 2)))
    for center in centers:
        cv2.circle(im0, center, 5, colors[int(cls)], 5)
```

Фигура 3.24 Откъс от detect.py, използващ уникалния номер (в случая 32) за обект от тип ‘топка за тенис на корт’ с цел извършване на действия при

наличие на такъв обект (в случая е направена имплементация за разчертаване на траекторията на тенис топката чрез установяване на центъра на кутията, която я обвива)

## 3.2 Визуални елементи (Electron)

**3.2.1 Главно приложение:** Първоначално в главното приложение се визуализира страница, на която потребителя има възможност да направи избор за броя на хората, които ще участват в тенис мач/тренировка. Структурата на тази страница е отличена във Фигура 3.25 и Фигура 3.26.

```
<head>
  <link href='index.css' rel='stylesheet' type='text/css'>
  <meta charset="UTF-8">
  <title>Hello World!</title>
  <meta http-equiv="Content-Security-Policy" content="script-src 'self' 'unsafe-inline'; " />
  <script src="page.js" type="text/javascript"></script>
</head>
```

Фигура 3.25 Съдържание на head тага на първата страница

```

<body>
  <div id="first">
    <h1>How many players will be participating?</h1>
    <div id="selection">
      <button id="one">
        <span>1</span>
      </button>
      <button id="two">
        <span>2</span>
      </button>
      <button id="four">
        <span>4</span>
      </button>
    </div>
  </div>
</body>

```

Фигура 3.26 Съдържание на body тага на първата страница

След като бъде направен съответен избор, няма пренасочване към друга страница, а чрез определени функции напълно се преобразява настоящата страница от гледна точка на съдържание и стилистика (Фигура 3.27, 3.28).

```

window.onload = function() {
  let pageHead = "<link href='page.css' rel='stylesheet' type='text/css'>" +
    "<meta charset='UTF-8'>" +
    "<title>Hello World!</title>" +
    "<meta http-equiv='Content-Security-Policy' content='script-src 'self' 'unsafe-inline';' />"
  let pageBody = "<div class='testbox' style='width: 100%; height: 100%'><h1>Player information</h1><form action='/'><hr> +
    "<input type='text' name='name' id='name' placeholder='First name' required/> +
    "<input type='text' name='name' id='name' placeholder='Last name' required/><hr> +
    "<div class='hand'> +
    "<input type='radio' value='None' id='left' name='hand' checked/><label for='left' class='radio'>Left-handed<
    "<input type='radio' value='None' id='right' name='hand'><label for='right' class='radio'>Right-handed</label>
    "</div><hr></form></div>"

```

Фигура 3.27 Ново съдържание на index.html

```
document.getElementById("first").style.display = "none";  
document.head.innerHTML = pageHead;  
document.body.innerHTML = pageBody + "<button id='detect'>Submit</button>";
```

Фигура 3.28 Чрез document.head.innerHTML и document.body.innerHTML се задава новата структура на страницата, като преди това старото съдържание се премахва чрез задаване на стойност none на display property-то на старата страница.

Оттук е осъществена връзка между Electron приложението и detect.py програмата. Това става чрез така наречения python-shell - npm пакет, който опростява изпълнението на Python програми през Node.js (Фигура 3.29).

```
const {PythonShell} = require('python-shell');  
  
PythonShell.run(['detect.py', null, function(err, results) {  
  if (err) throw err;  
  console.log(`${results}`);  
}])
```

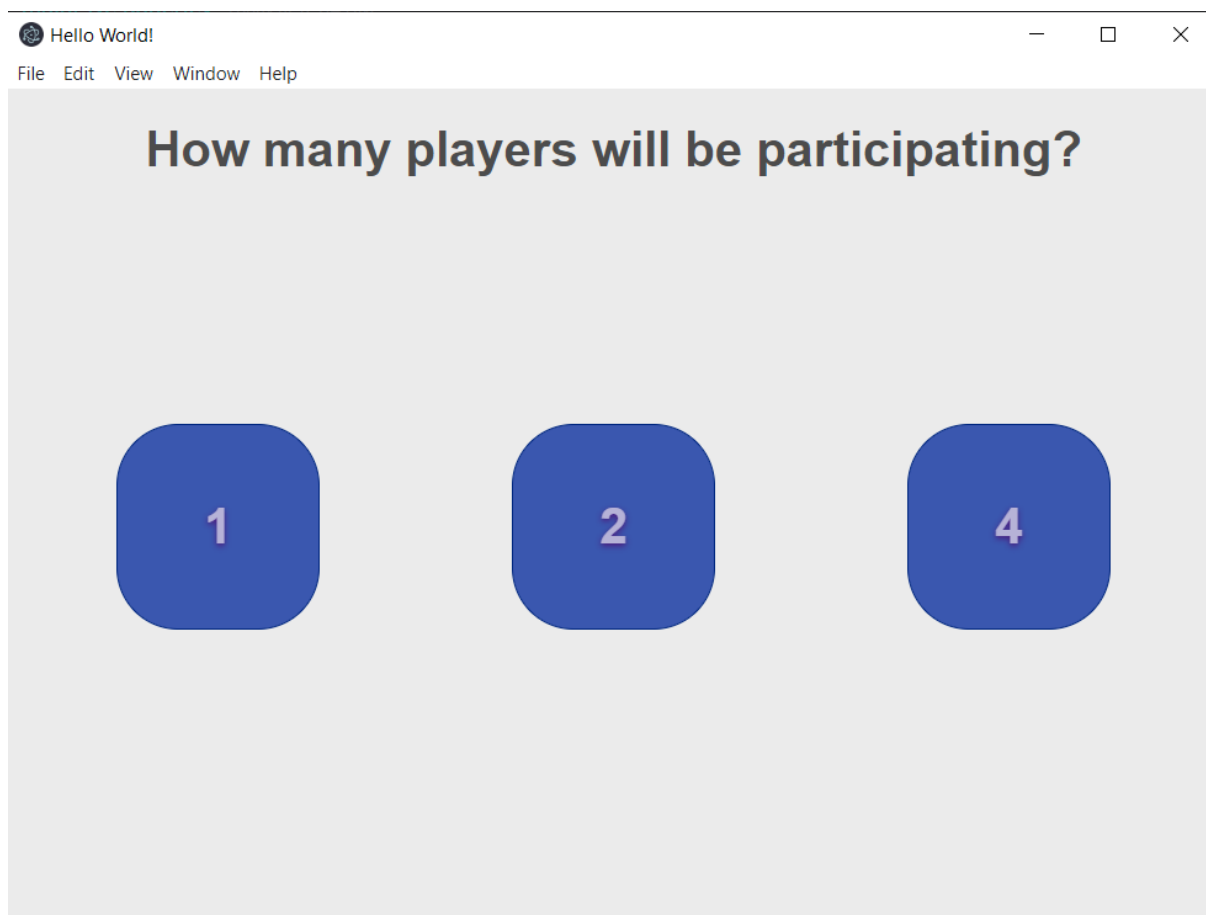
Фигура 3.29 Употреба на python-shell с цел връзка между Electron приложението и detect.py

# Четвърта

# глава

Ръководство за потребителя





Фигура 4.1 Първоначален изглед при зареждане на Electron приложението

При стартиране на приложението потребителят се озовава на страницата, показана на Фигура 4.1. Там той има няколко опции и в зависимост от тази, която подбере, по-нататъшната визуализация ще се види по определен начин (три избора съответстват на три различни визуализации - Фигура 4.2, 4.3, 4.4).

Hello World!

File Edit View Window Help

## Player information

First name

Last name

☒ Left-handed ☐ Right-handed

Submit

Фигура 4.2 Визуализация за един избран участващ

Hello World!

File Edit View Window Help

Player information

First name

Last name

☒ Left-handed ☐ Right-handed

Submit

Фигура 4.3 Визуализация за двама избрани участващи

The image shows a web browser window with the title 'Hello World!'. The browser's menu bar includes 'File', 'Edit', 'View', 'Window', and 'Help'. The main content area displays four identical 'Player information' forms arranged in a 2x2 grid. Each form has a title 'Player information' at the top. Below the title are two input fields: 'First name' and 'Last name'. Under these fields are two radio buttons: 'Left-handed' (which is selected, indicated by a blue checkmark) and 'Right-handed'. To the right of the four forms is a large blue vertical button labeled 'Submit'.

Фигура 4.4 Визуализация за четирима избрани участващи

Пред потребителят е представен определен брой формуляри, за чието попълване трябва да се погрижи. След като е попълнена необходимата информация, трябва да бъде натиснат бутона Submit за потвърждение и оттам се задейства програмата detect.py при определена физическа конфигурация (камери, позиции и така нататък). Започва заснемане в реално време, получените кадри се обработват и от тях се извлича

определена информация, която подлежи на разглеждане и оттам на визуализация при приключване на същинския процес.

# Заклучение

През времето, в което се развива и изследва, изкуственият интелект е създал и продължава да създава все по-интересни и сложни предпоставки за разработка на системи от по-високо ниво. Разработката на тази система дотам, докъдето е в момента, се оказва доста предизвикателна и наистина се опираше през доста голяма част от времето. Всичко, което е постигнато на този етап, не е напразно - трудностите, които видно не спираха да се появяват, изградиха упоритост и помогнаха в натрупването на нови знания. На фона на това обаче има още изисквания, които трябва да се подобрят и да се реализират цялостно. Винаги може по-добре, в този случай обаче може доста по-добре и до такава степен на реализация ще се добута:

- Визуалната част на системата предстои да се допълни и евентуално да се подобри.
- Главните изисквания ще бъдат довършени.
- Ще се създадат по-добри предпоставки за осъществяване на обработка на всеки вид информация, която се обменя в рамките на системата.
- Системата ще се конфигурира по адекватен начин така, както е възнамерено да бъде направена.

### Използвана литература:

1. SSD, <https://arxiv.org/abs/1512.02325>
2. YOLO v1,  
<https://towardsdatascience.com/evolution-of-yolo-yolo-version-1-afb8af302bd2>
3. YOLO v2, <https://arxiv.org/abs/1612.08242>
4. YOLO v3,  
<https://towardsdatascience.com/yolo-v3-object-detection-53fb7d3bfe6b>
5. YOLO v4,  
<https://towardsdatascience.com/yolo-v4-optimal-speed-accuracy-for-object-detection-79896ed47b50>
6. YOLO v5, <https://github.com/ultralytics/yolov5>
7. Python, <https://www.python.org/>
8. Haskell, <https://www.haskell.org/>
9. LISP, <https://www.tutorialspoint.com/lisp/index.htm>
10. R, <https://www.r-project.org/>
11. Julia, <https://julialang.org/>
12. Spyder, <https://www.spyder-ide.org/>
13. Visual Studio Code. <https://code.visualstudio.com/>
14. PyCharm, <https://www.jetbrains.com/pycharm/>

15. TrackNet,  
<https://nol.cs.nctu.edu.tw:234/open-source/TrackNet/tree/master>;  
<https://inoliao.github.io/CoachAI/>
16. Hawk-Eye, <https://www.hawkeyeinnovations.com/sports/tennis>
17. OpenCV, <https://opencv.org/>
18. NumPy, <https://numpy.org/>
19. Matplotlib, <https://matplotlib.org/>
20. PyTorch, <https://pytorch.org/>
21. Electron, <https://www.electronjs.org/docs>



## Съдържание

Увод	4
Първа глава	7
1.1 Технологии за създаване на система за анализ чрез object detection.	8
<b>1.1.1 SSD (Single Shot Detection).</b>	8
<b>1.1.2 Yolo (You Only Look Once).</b>	9
1.2 Програмни езици, използвани за AI и математически анализ.	12
1.3 Подходящи развойни среди за машинно обучение.	14
1.4 Съществуващи разработки на системи за следене на обекти в тениса.	15
Втора глава	18
2.1 Функционални изисквания към система за анализиране на тенис мач/тренировка:	19
2.2 Избрани програмни средства и развойна среда съобразени с проекта:	19
<b>2.2.1 Visual Studio Code</b>	19
<b>2.2.2 Python</b>	22
<b>2.2.3 Electron</b>	31
Трета глава	35
3.1 Установяване и обработка на информация	36
<b>3.1.1 court.py</b>	36
<b>3.1.2 detect.py</b>	39
3.2 Визуални елементи	45
<b>3.2.1 Главно приложение</b>	45
Четвърта глава	48
Заклучение	53
Използвана литература:	55
	57