**ТЕХНОЛОГИЧНО УЧИЛИЩЕ ЕЛЕКТРОННИ СИСТЕМИ**

**към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ**

**ДИПЛОМНА РАБОТА**

Тема: Роботизирана кола с вградена система и дистанционно

управление чрез смартфон

Дипломант: Научен ръководител:

*Георги Низамски инж. Георги Константинов*

СОФИЯ

2018

**TЕХНОЛОГИЧНО УЧИЛИЩЕ**



**ЕЛЕКТРОННИ СИСТЕМИ**

**към ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ**

Дата на заданието: 06.11.2017 г. Утвърждавам:..............................

Дата на предаване: 06.02.2018 г. /проф. д-р инж. Т. Василева/

**ЗАДАНИЕ**

**за дипломна работа**

на ученика Георги Пламенов Низамски 12 Б клас

1.Тема: Роботизирана кола с вградена система и дистанционно управление чрез смартфон

2.Изисквания:

- Безжична WiFi връзка между устройствата

- Android приложение, с което да се извършва управлението на количката

- Камера, с която да се вижда в Android приложението случващото се пред количката

3.Съдържание 3.1 Обзор

3.2 Същинска част

3.3 Приложение

Дипломант : ..............................

Ръководител:..........................................

/инж. Георги Константинов/

Директор:................................................

/ доц. д-р инж. Ст. Стефанова /

УВОД

В глобален мащаб, с напредването на електрониката и технологиите, нараства производството и използването на роботизирани системи (роботи) в различни области от съвременния свят. Пример за това е един основен дял от промишлеността, а именно производството на автомобили. Роботиката, като немалка част от електроинженерството, се състой в проектирането и конструирането на роботи, както и написването на управляващ софтуер, за техния контрол. Най-често роботите се използват за изпълнението на повтарящи се задачи. В допълнение, те са полезни в среда, която е опасна за живота на хората, като например при обезвреждане на бомби, подводно проучване на голяма дълбочина [1]. Някои роботи могат да действат, според собствената си способност за вземане на решения, осигурена от технологията за изкуствен интелект [1]. Все повече хора започват да припознават създаването на роботи, като свое хоби.

Вградените системи са една основна част от контролния механизъм на един робот. Обикновено са част от друга, по-голяма система. Характеризират се с това, че работят самостоятелно и изключително надеждно. Друго важно условие за работата на този вид системи е изпълнението на задачите, в реално време. По тази характеристика се разделят на Soft, Firm и Hard Real Time системи. При Hard Real Time системите се счита, че всеки пропуснат краен срок за изпълнение на дадена задача, е системна грешка [2]. При Firm Real Time системите се допуска пропускането на срокове, стига да са през някакво времево разстояние, но по този начин производителността може да падне до нула [2]. При Soft Real Time системите позволяват често пропускане на крайни срокове, докато задачите продължават да имат стойност [2].

Основно предназначение на вградените системи е да изпълняват ограничен набор от функции, най-често една или две.

Целта на дипломната работа е да се създаде прототип на робот, който може да бъде в разнообразни индустрии и ежедневието ни. Ще бъдат реализирани следните изисквания:

* Комуникацията между управляващия софтуер (приложението) и роботизираната система да е безжична (WiFi)
* Разработване на приложение, базирано на операционната система Android, чрез което да бъде управлявана колата
* Подмяна на управляващия хардуер на радио-управляема кола, с микроконтролерна платка Raspberry Pi 3
* Поставяне на камера, свързана към микроконтролера, чрез която да се вижда случващото се пред колата, по време на движение

ПЪРВА ГЛАВА

ОБЗОР НА СЪЩЕСТВУВАЩИТЕ ПОДОБНИ СИСТЕМИ, КАКТО И МЕТОДИТЕ, ТЕХНОЛОГИИТЕ И РАЗВОЙНИТЕ СРЕДИ ЗА ТЯХНАТА РАЗРАБОТКА. ПРОУЧВАНЕ НА ЕЛЕМЕНТНА БАЗА

1.1 Обзор на съществуващи подобни системи

Роботизираните системи намират широко приложение във военната индустрия. Инвестират се изключително много ресурси и време в проектиране и производство на роботи. Всеки произведен модел робот има специфични характеристики – размер, тегло, проходимост, водоустойчивост, обхват на сигнала и други. Основната задача на този тип роботи е да осигурят защитата на своя оператор.

Военните роботизирани системи се разделят основно в две групи UGV абревиатурата идва от - Unmanned Ground Vehicle, на български Безпилотен Наземен Апарат. Втората група са UAV или Unmanned Aerial Vehicle, на български Безпилотен Летателен Апарат. QinetiQ North America е световен лидер в разработването и доставянето на безпилотни системи с над 4500 робота, разположени в цял свят [3] . В тази проучвателна глава, от дипломната работа ще бъдат разгледани два наземни безпилотни робота (UGV) - QinetiQ North America Dragon Runner 10 и Dragon Runner 20 [4].

1.1.1 Робот QinetiQ North America Dragon Runner 10

Семейството роботи Dragon Runner, проектирани от фирмата QinetiQ North America, предоставят световна класа в подкрепа на военните части. Този тип роботи са лесно транпортируеми и позволяват операторите си, да ги пренасят на най-опасните условия и терени, съществуващи на Земята [4].

Според размерите си, Dragon Runner 10 (DR - 10) е най-малкия безпилотен наземен робот. Затова той спада към групата на MUGV (Micro Unmanned Groud Vehicle). Този робот е лек, предлага защита и поддръжка на малки единици. Основното му предназначение е да осигури осведоменост, чрез процес на разузнаване, докато неговия оператор стой на безопасно разстояние от конфликтната точка [5].

Със своето приблизително тегло от 5 килограма DR – 10 може да бъде носен в ръка, както лесно да се транспортира в раница – дори, в някой случай може да се наложи да бъде хвърлен. Този малък, но устойчив робот се управлява от лесно за използване дистанционно управление (контролер). Работата и моделът на този контролер ще бъдат разгледани по-нататък, по време на проучването. Дизайнът му позволява да изпълнява задачите си дори в най-предизвикателната обстановка и неблагоприятен климат [5]. Изображение на робота се вижда на Фиг. 1.1



***Фиг 1.1 QinetiQ North America Dragon Runner 10***

В допълнение, роботът е оборудван с дневни и нощни сензори. Чрез тези сензори той влиза в ролята на “очи” на съответната военна част, докато хората стоят на безопасно разстояние. Използва се още, за поставяне на модули за разоръжаване на импровизирано взривно устройство. Независимо дали под кола или в канализацията, DR – 10 има възможността да отиде там, където един човек не би могъл [5].

*Спецификации на DR – 10 [5]:*

* Тегло: 10 паунда ≈ 5 килограма
* Размери: 15.5’’ (40 см) х 13.8’’ (35 см) х 6’’ (15.2 см) (Д х Ш х В)
* Камери: Предна и задна, Ден и нощ
* Вграден микрофон
* Скорост на придвижване: 4 mph (≈ 6.4 kmh)
* Обхват на радиовръзка: 650+ метра
* Устойчивост: IP65
* Контролер: Съвместим с всички контролери на QinetiQ
* Издръжливост: 2-3 часа (в зависимост от вида на мисията)

1.1.2 Робот QinetiQ North America Dragon Runner 20

Вторият робот, който ще бъде на вниманието на това проучване е следващия безпилотен апарат от семейството Dragon Runner на фирма QinetiQ NA. За разлика от по-малкия по физически размери DR – 10 този робот спада към друга категория роботи, именно SUGV (Small Unmanned Ground Vehicle). Първоначално конструиран за морския корпус на САЩ, базовият модел на този робот тежи 20 паунда (приблизително 9 килограма) [4]. Въпреки, че на размери не е по-голям от една детска играчка, той обединява набор от средства, които да допринесат към успеха на дадена мисия [4]. Изображение на този робот се вижда на Фиг. 1.2



***Фиг 1.2 QinetiQ North America Dragon Runner 20***

Американският робот Dragon Runner 20, за по-кратко наричан DR – 20, е лек, лесно преносим, както и високо специализиран. Основните задачи, които трябва да изпълнява са: ситуационна осведоменост, чрез процес на разузнаване (подобно на DR - 10). Изключително подходящ е за разоръжаване, обезвреждане на боеприпаси, както и други военни операции [6]. Оборудването, с камери и сензори, предоставят на оператора, да вижда какво се случва пред робота, като стой на безопасно разстояние [6].

Роботът е достатъчно малък, за да е възможно пренасянето в ръка или в раница. В същото време има механичните възможности да повдигне предмет до 10 паунда или приблизително 5 килограма. DR – 20 притежава роботизирана ръка, с въртящо се рамо и китка, като движенията се извършват прецизно и сръчно. Притежава възможности за наблюдение с дневни и нощни камери, както и подслушване с микрофон [6].

*Спецификации на DR – 20 [6]:*

* Тегло: 20 паунда ≈ 9 килограма
* Размери: 16.6’’ (42 см) х 12.2’’ (31 см) х 6’’ (15.2 см) (Д х Ш х В)
* Капацитет за повдигане на предмети: до 10 паунда (5 кг)
* Камери: Предна и задна, мащабиране/накланяне
* Подобрена работа с контролера
* Вграден микрофон
* Контролер: Съвместим с контролери на QinetiQ

Към основната версия на този модел роботи, може да се прикачат допълнителни модули и устройства, с цел по-добра производителност и успеваемост. Модулите, които могат да бъдат прикачени към корпуса на DR – 20 са [6]:

* + Подобряване на веригите
  + Добавяне на многоосно манипулаторно рамо
  + Добавяне на серво-камера, с различни възможности, като мащабиране/накланяне
  + Двупосочна комуникация
  + RSTA Payload (Reconnaissance, surveillance, and target acquisition) - Получаване на разузнавателна, наблюдателна и целева информация

1.1.3 Tactical Robot Controller

Роботизираните системи биват управлявани от хора, наречени оператори. Контролирането на роботи, като споменатите по-горе, се извършва от разстояние. Комуникацията между контролера и робота, е безжична – обичайно се използва радиочестотно управление, като информацията се предава чрез радиовълни.

В тази част от проучването ще бъде разгледан Тактическият роботизиран контролер (Tactical Robot Controller - TRC) на QinetiQ. Изображение на контролера се вижда на Фиг 1.3

****

***Фиг 1.3 QinetiQ Tactical Robot Controller***

Този контролер е лек, удобен за носене, като дава възможност на оператора да контролира различни модели на безпилотни наземни роботи (UGV), безпилотни летателни апарати (UAV) и наземни датчици. Операторът прикача комуникационния пакет към системата, която иска да контролира [8]. Контролерът има достатъчно технологични възможности. Притежава 1.2 или 1.8 Ghz двуядрен процесор, USB/Ethernet портове. Предоставя Bluetooth/WiFi/GPS комуникация, в допълнение има достатъчно оперативна памет 4 GB, и 6.5’’ дисплей [8].

1.2 Преглед на хардуерни платформи

До момента в тази глава проучването беше насочено към строго специализирани военни роботи. Но, освен този тип роботи, съществуват и множество хоби-роботи, който се конструират с помощта на микроконтролерни платки. Най-популярните такива са Arduino и Raspberry Pi.

В рамките на тази част от проучвателната глава ще бъдат разгледани две развойни среди – Arduino IDE и IntelliJ, който се използват при разработката на софтуер, за управление на тези микроконтролерни платки.

1.2.1 Arduino

Arduino е платформа, с отворен код, базирана на съвкупността от хардуер и софтуер [11]. Хардуерът се изразява в разработката на развойни платки, който се разпространяват под Creative Commons лиценза [9].

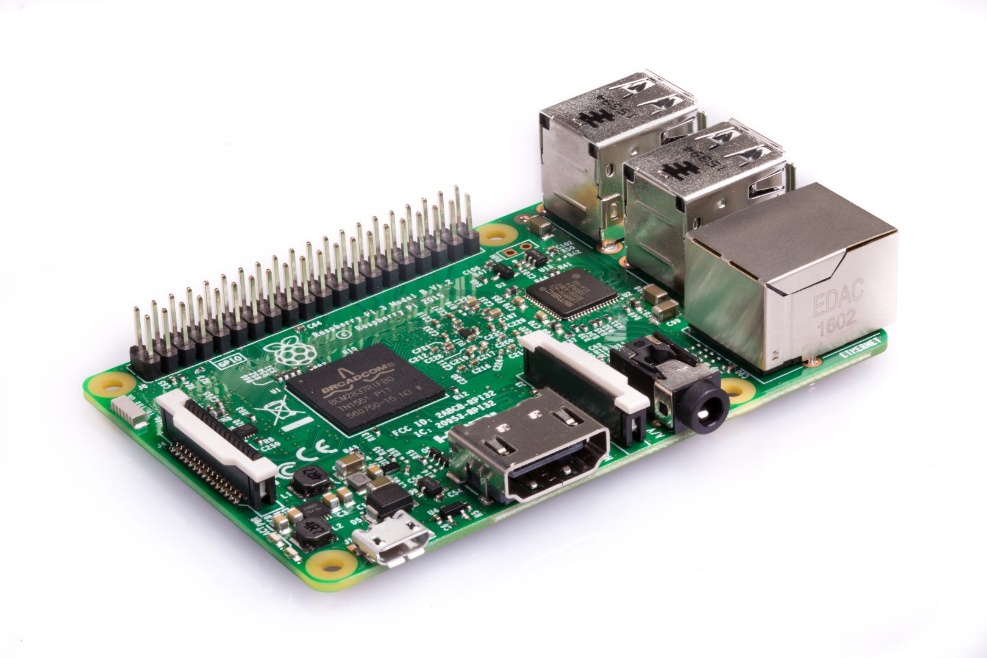
Ядрото на тези платки е 8-битов Atmel AVR (ATmega328) микроконтролер, с различно количество флаш памет, входно-изходни пинове и други характеристики. Съществува и Ардуино платка, която се състой от 32-битов Atmel процесор, базиран на ARM Cortex M3 [9]. Микроконтролерите Arduino съдържат boot loader, който значително улеснява качването на програми върху флаш паметта на съответното устройство. Програмният код достига до паметта, посредством серийна връзка. В това отношение захранването нужно за нормалната работа на Ардуино платките е между 7 - 12 V. Други физически и технически параметри на този тип микроконтролерни платки, като Ардуино Uno и Arduino Diecimila, са: осигуряват 14 цифрови и 6 аналогови входно-изходни пина.

***Фиг 1.4 Arduino Uno***

На Фиг 1.4 се вижда Arduino Uno – популярен модел микроконтролерна платка. В допълнение са разработени Ардуино съвместими печатни платки, които се използват за разширение на дадена система, наричат се шийлдове. Те добавят функционалност, като например осигуряват управление на мотори, Bluetooth, WiFi, GPS комуникация. Свързването към „главната“ платка е посредством входно-изходните пинове [9].

1.2.2 Raspberry Pi

Raspberry Pi представлява серия от едночипови компютърни системи, проектирани и разработени в Обединеното Кралство, от Raspberry Pi Foundation. Основното им предназначение е да популяризират електрониката и роботиката [10]. Към продуктовия пакет не са включени периферни устройства, но е осигурена възможност за тяхното свързване.

**** Последна генерация платки са Raspberry Pi 3, който променят изцяло визията за досегашните компютърни системи, от този тип. Raspberry Pi 3 притежава четириядрен ARM Cortex A53 процесор, който е около 10 пъти по-производителен от първата версия Raspberry Pi 1 и приблизително 80% по-ефективен, в изпълнението на задачи, от предшественика си Raspberry Pi 2 [10].

***Фиг 1.5 Raspberry Pi 3 Model B***

***Фиг 1.5 Raspberry Pi 3***

На Фиг 1.5се вижда последният модел на Raspberry Pi Foundation – Raspberry Pi 3 Model B. Моделът притежава четириядрен процесор ARM Cortex A53, с 1.2 гигахерца тактова честота, 1 гигабайт RAM, 40 входно-изходни пинa, както и различни портове – HDMI, 4xUSB, Ethernet, CSI (Camera Serial Interface) порт за камера и други [12]. Но, най-съществената разлика с предходния модел е, че този е снабден с вграден WiFi модул.

Заради производителността и възможностите на този модел Pi платката поддържа голям брой от операционни системи – от повечето Linux дистрибуции до Microsoft Windows 10. Официалната операционна система, препоръчана и разработена от Raspberry Pi Foundation се нарича Raspbian. Това е безплатна операционна система, базирана на Debian, която е силно оптимизирана за хардуера на Raspberry Pi. Съдържа 35 000 пакета – пре-компилиран софтуер, пакетиран и готов за лесно инсталиране на съответния Rapsberry Pi модел [13].

Най-популярният език за разработка върху тези платки е Python. Но, въпреки това те поддържат и голям брой други програмни езици и технологии, като Java, Ruby, C, C++.

1.3 Преглед на известни развойни среди при разработката на софтуер за вградени системи

В настоящата документация ще бъдат разгледани няколко известни развойни среди при разработката на софтуер за различни хардуерни платформи – Arduino IDE, IntelliJ IDEA и Eclipse.

1.3.1 Arduino IDE

Тъй като в основата на Ардуино платките са микроконтролери на Atmel, фирмата се е погрижила да осигури среди за разработка на софтуер, а именно AVR Studio и Atmel Studio. Но, обект на това проучване е интегрирана среда за разработка, която е проект на Ардуино. Тя е мулти-платформено приложение, написано на Java, което се базира на интегрираните среди за разработка на езика “Processing” и платформата „Wiring“ [9].

Редакторът на код има базови функции като копиране и поставяне, търсене и заменяне на текст. Други функционалности, които предоставя този редактор са: променяне на цвета на кода, в зависимост от категорията на термините - „Syntax Highlighting“, както и подчертаване или оцветяване на съвпадащи скоби - “Brace Matching”. Дизайнът на приложението е направен лесен за използване, пример затова е изключително лесният начин за компилиране и качване на кода до хардуерната платформа, само чрез натискане на един бутон. Програма написана на Arduino IDE е прието да се нарича „sketch“ (скица). Във файловата система се записва като текстов файл, но с разширение .ino.

Интегрираната среда за разработка на Ардуино поддържа програмните езици C и C++. За да бъде успешно компилиран, написаният код изисква наличието на две функции *setup()* и *loop()*.

*Setup()* функцията се извиква веднъж след започването на изпълнението на програмата. Нейното основно предназначение е да инициализира променливи и да определя режимите вход/изход на пиновете.

*Loop()* функцията се извиква многократно по време на изпълнението на програмата, но само при условие че, *setup()* функцията е приключила успешно своето изпълнение. *Loop()* има за цел да управлява платката, до нейното изключване или рестартиране. Процесът на компилация на сорс кода е специфичен. Първо Ардуино средата добавя ново разширение .cpp на всеки файл завърващ с .ino, като по-този начин превръща „скицата“ в C++ програма. Към началото на файла се добавя #include <Arduino.h> заглавен файл, които включва всички дефиниции, необходими за нормалната работа на Ардуино ядрото. След това новосъздадената C++ програма се предава на AVR Libc компилатор, който превръща четимия код в инстукции или обектни файлове [14]. И последната стъпка е добавянето на стандартните Ардуино библиотеки, като краен резултат се получава файл, който е в шестнадесетична бройна система и трябва да бъде записан в паметта на Ардуино платката.

1.3.2 IntelliJ IDEA

IntelliJ IDEA представлява интегрирана среда за разработка (IDE) създадена от фирмата JetBrains. Базира се на IntelliJ платформата, като първата версия на приложението е пусната през 2001 година и бързо се налага на пазара, заради своите възможности, които се изразяват в разширено навигиране по кода и лесни начини за рефакториране (процес, при който се променя структурата на кода, но не и начинът на работа). Самата среда за разработка е написана на Java. Предоставя набор от средства, които улесняват работата на програмистите. Някой от тях са „Smart Completion“ – това е процес, при който се дава списък с най-подходящите символи, които могат да бъдат приложени в съответно парче от кода, „Detecting duplicates“ - процес, който открива фрагменти от кода, които съвпадат [15].

Към момента съществуват две версии на приложението – IntelliJ IDEA Community и Ultimate Edition. Community версията е с отворен код, под лиценза Apache 2.0. Поддържа основен пакет от технологии и програмни езици, като Java, Groovy, Scala и инструменти – Maven и Gradle. Ultimate версията от своя страна е платена и съдържа пълен пакет от софтуерни технологии.

IntelliJ е платформа за изграждане на интегрирани среди за разработка, която е основа на всички продукти на JetBrains. Тя предоставя набор от инструменти за успешното разработване на този тип софтуерни продукти. Някои от тях са виртуална файлова система, текстов редактор, дебъгър и потребителски интерфейс [17].

1.3.3 Eclipse

Eclipse е общност, чиито проекти са насочени към изграждане на една гъвкава платформа за разработка. Всички продукти, както и платформата са с отворен код. Това позволява на много хора и фирми да изграждат свои приложения и инструменти, които да се базират на нея.

Eclipse предоставя може би най-популярната интегрирана среда за разработка, която се използва предимно за изграждане на Java приложения.

Поради това, че Eclipse общността е изключително мащабна са създадени множество приставки, които са способни да разширят основната функционалност на средата. Една много популярна приставка за Eclipse е тази, която осигурява възможността за разработка на приложения, базирани на Android операционна система (на английски - Android Development Tools).

Eclipse е написан на обектно-ориентирания език Java. В действителност същинската част на средата е много малка. Всички функционалности, които има се поддържат от приставки. Важно е да се отбележи, че всяка приставка се зарежда в паметта само когато е необходима. По този начин не се заема излишна памет и други ресурси на компютърната система.

Основно предимство на Eclipse пред IntelliJ е че, Eclipse е безплатен, проект с отворен код и чрез използването на приставки, може да се постигне същата производителност и поддръжка.

Разработката на Eclipse проектите се извършва под крилото на Eclipse Foundation, неправителствена организация, която е подкрепяна от софтуерни фирми по цял свят. Всички продукти създадени от Eclipse Foundation се разпространяват от специално създаден лиценз – Eclipse Public License (EPL).

1.4 Преглед на известните операционни системи за мобилни устройства

1.4.1 iOS

iOS е една от най-популярните операционни системи. Разработена е от компанията Apple Inc. Първата версия на тази операционна система се появява заедно с iPhone и се нарича iPhone OS 1. Както се вижда от името на операционната система, първоначално е била създадена само за популярния смартфон на компанията (iPhone). По-късно са проектирани и създадени други мобилни устройства, като - iPad, iPod и Apple TV и именно тази операционна система е вградена в тях.

Всяка година от компанията пускат нова версия на системата. Текущата версия към януари 2018 година е iOS 11.2.5. По същество потребителският интерфейс на операционната система се състой от плъзгачи, ключове, бутони подобно на другата популярна операционна система Android.

В действителност iOS е разработена на основата на друга операционна система, притежание на Apple Inc. – Mac OS X, която от своя страна е базирана на Unix. Развитието на iOS е основно заради създаването на много приложения, които добавят функционалност към ядрото на системата. Тези приложения могат да бъдат свалени и инсталирани от официален интернет магазин поддържан от Apple Inc., именно App Store. Приложенията, които биват разработвани и качвани в този магазин са предназначени само устройствата, който са с iOS.

1.4.2 Android

Андроид (на английски: Android) е към момента най-популярната операционна система за мобилни устройства в света. Създадена е от компанията Android Inc., a по-късно е купена от софтуерния гигант Google Inc. Базира се на модифицирана версия на Linux ядро. Компанията Google развива няколко варианта на системата Android TV за телевизори, Android Wear за ръчни часовници и Android Auto за автомобили. Към момента последната версия, предоставена на потребителя е Android 8.1 Oreo.

Към развитието на операционната система допринасят основно приложенията, които се разработват с цел да се разшири функционалността. Тези приложения могат да бъдат свалени от специален интернет магазин, собственост на Google – Google Play Магазин. Като програмни езици за написването на приложенията се използва главно Java. Потребителският интерфейс на тази операционна система се основава на принципа за директна манипулация, използвайки докосвания по екрана на устройството, за да се манипулират обекти.

Операционната система Android се разработва под лиценза – Apache License 2.0. Пакетният мениджър на Android (на английски: Android Package), се използва за разпространение и инсталиране на приложенията.

Мобилните устройства се захранват от батерия, ето защо и Google, и Apple се стремят операционните им системи да управляват процесите си така, че да консумират минимално количество енергия. Например при Aндроид, управлението на процеси се извършва по-същия начин, както работят приставките в Eclipse – когато едно приложение не се използва, системата прекратява временно работата му, така че при нужда да може лесно да го възстанови.

ВТОРА ГЛАВА

ОБЩИ ИЗИСКВАНИЯ И ПРОЕКТИРАНЕ НА ПРОГРАМНИЯ ПРОДУКТ. ИЗБОР НА ПРОГРАМНИ ЕЗИЦИ И СРЕДИ ЗА РАЗРАБОТКА

2.1 Общи изисквания към програмния продукт

Целта на дипломната работа е да се разработи роботизирана кола, която да се управлява чрез мобилно приложение базирано на операционната система Android. Изискванията към продукта са:

* + Движението на колата да се извършва в следните посоки: напред, назад, наляво и надясно. Като може да се комбинират напред и надясно, напред и наляво, назад и надясно, назад и наляво.
  + Поставяне на камера върху корпуса на колата, която да е свързана към съответния хардуер, като по този начин да се вижда случващото се по време да движение
  + Мобилното приложение трябва да може да свързва съответния смартфон със колата, посредством безжична връзка
  + Изпращане на HTTP заявки от приложението до сървър, който да ги обработва и по този начин да извършва управлението на колата

2.2 Избор на език за програмиране на управляващия софтуер и мобилното приложение

В настоящия проект избраният език за програмиране е Java. Java съвременен обектно-ориентиран език. Обектно-ориентираният стил за писане на код, значително опростява разработването на сложни софтуерни системи. На практика на Java се създават най-различни видове приложения. Подобно на езици като C#, Java е език от високо ниво. Програмите написани на този език са изградени изцяло от класове, като на практика липсват глобални методи и променливи. Спрямо предшественика си C++, в Java е премахнато множественото наследяване и предефинирането на оператори.

Най-голямото предимство на Java и това което я отличава от останалите програми езици и платформи е, че е лесно преносима върху всякакви хардуерни и софтуерни системи. Тези конфигурации трябва да имат инсталиран Java Runtime Environment (JRE), който съдържа специална Java виртуална машина (JVM). Процесът на компилиране на Java кода е изключително специфичен, тъй като не се компилира до машинен код, а до така наречения байт код (на английски „bytecode“), който може да бъде изпълнен на всяка Java виртуална машина.

Друга част от платформата на Java е Java Development Kit (JDK) - това е пакет от инструменти, който позволява разработването на програми, както и тяхната компилация. Много важно предимство, което се предоставя от виртуалната машина на Java е автоматичното управление на паметта, като динамичната памет се заделя в специално пространство, наречено „heap“.

Референции (Препратки) към обектите в динамичната памет, се намират в друга памет – „стек“ („stack“). Процесът по освобождаване на паметта е задължение на така наречения „garbage collector“. Когато към един обект, няма повече референции и неговата памет е ненужна за програмата, тя бива освободена. В Java е невъзможна външната намеса при управлението на паметта, затова в езика не се използват указатели. На практика грешките при този процес за сведени до минимум.

Естествено Java си има и някои недостатъци. Програмите написани на този език консумират повече ресурси откъм процесорно време и оперативна памет, най-вече заради процесите, които се изпълняват от виртуалната машина.

2.3 Избор на развойна среда при разработката на управляващия софтуер

В контекста на дипломната работа „управляващ“ е този софтуер, чрез който се контролира платката, поставена на корпуса на колата. За развойна среда на това приложение е избран IntelliJ IDEA, базиран на IntelliJ платформата. За разработката на приложението за текущия проект е използвана Ultimate версията на IntelliJ, тъй като тя предоставя по-голям пакет от технологии и инструменти. Eclipse и IntelliJ използват приставки за разширяване на функционалността си. По време на разработката на приложението, което контролира развойния хардуер, беше приложен процес наречен дистанционно отстраняване на грешки (на английски: remote debugging). В този контекст беше използвана приставка, която да улесни този процес като с натискането на бутона „Run“ в IntelliJ, кода се пакетира и по SSH връзка се качва върху хардуера, където се изпълнява. По този начин се избягва ръчното създаване на .jar файл, неговото качване и изпълнение, също така се отстраняването на грешки става значително по-лесен процес.

Друга основна причина и може би най-важна, заради която беше избран IntelliJ е, че основната среда за разработка на Android приложения -Android Studio е базирана на IntelliJ платформата. Това улеснява значително разработката на цялостния проект.

2.4 Избор на развойна среда при разработката на мобилното приложение

Мобилното приложение играе ролята на „дистанционното управление“, с което се управлява колата от настоящия проект. Развойната среда за неговата разработка беше избрана да е Android Studio.

Android Studio представлява интегрирана среда за разработка на приложения базирани на операционната система Android. Средата е създадена от софтуерната компания Google и първата версия е анонсирана през май 2013 година. Самото приложение се базира на IntelliJ платформата, на фирма JetBrains. В Studio са включени инструменти, които да улесняват работата на програмистите. Един много популярен инструмент, при разработката на Android приложения е Gradle. Този инструмент се използва за по-лесно изграждане (на английски: build) на проекти. Предимството на Gradle се изразява в това, че е базиран на езика Groovy, като по този начин се избягва писането на XML, който се използва от Apache Maven.

Много важно качество на Android Studio е, че съдържа в себе си емулатори, които позволяват написаните приложения да се тестват без използването на външно устройство (смартфон).

Програмният редактор в средата е много подобен на този на IntelliJ IDEA, като разполага с различни интелигентни системи, който помагат за усъвършенстването на програмния код.

2.5 Избор на основен развоен хардуер

Основният развоен хардуер избран за този проект е Raspberry Pi 3 Model B v1.2. Представлява едночипова компютърна система, създадена от английската компания Raspberry Pi Foundation, представена на пазара през Февруари 2016 година. Създадена е с цел популяризирането на електрониката и роботиката, първоначално във Великобритания, а после по света. Основната причина, заради която, за този проект беше избрана тази компютърна система е, че има значително по-големи възможности в сравнение с други микроконтролерни платки, например Arduino Uno. В конкретиката на проекта беше сметнато че, Uno не притежава необходимите изчислителни възможности и мощност, за да поеме захранването и движението на колата, както и поддръжката видео връзка през камерата. Това не означава, че Arduino Uno е неефективна хардуерна платформа, просто е предназначена за изпълнението на друг тип задачи и създаване на различни проекти.

Платката поддържа множество операционни системи - повечето Linux базирани както и Microsoft Windows 10. Raspberry Pi Foundation препоръчват тяхната операционна система Raspbian, която е базирана на Debian и е оптимизирана за работа с Pi хардуера.

Най-широкоразпространеният език за програмиране върху тези платки е Python, но естествено се поддържат и набор от други езици - Java, Ruby, C, C++.

В допълнение Raspberry Pi 3 е една изключително лесна за използване платка, която за цената си, предоставя много възможности за реализация в различни проекти.

2.6 Spring Framework и проектът Spring Boot

Framework или „фреймуърк“ е платформа, която дава основа на разработчиците при създаването на приложения за съответна програмна технология или език. Тя най-често съдържа в себе си методи или класове, които могат да улеснят работата на един програмист. Може би най-популярният фреймуърк за Java е Spring. Първата версия на Spring е анонсирана през 2002 от фирма Pivotal Software, като платформата е с отворен код и до днес.

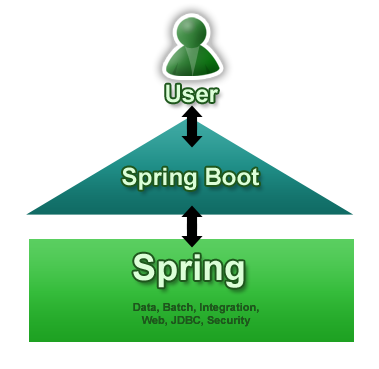
Тя предоставя програмен и конфигурационен модел за модерни корпоративни приложения, които са базирани на Java. Ключов елемент представлява поддръжката, която Spring осигурява на ниво приложение [18].

Освобождава хората от грижата за конфигурирането на някой компоненти от приложението, като тези процеси се изпълняват скрито и по този начин екипите се съсредоточават към усъвършенстване на бизнес логиката [18].

Изключително важно предимство на Spring е, че той използва така наречения процес – „инжектиране на зависимости“ (на английски Dependency Injection). Това е начин за създаване на обекти, които зависят от други обекти. Системата за инжектиране на зависимости доставя зависимите обекти (наричани зависимости), когато създава инстанция на новия обект [19]. Обектите, които формират структурата на дадено Spring приложение се наричат, на английски „bean”. Тези обекти от своя страна се инстанцират от контейнера за инжектиране на зависимости. Процесът по създаване на един такъв обект е следния: обектите сами определят своите зависимости, чрез конструктори или методи. След като са вече определени, контейнерът „инжектира“ тези зависимости, когато създава един инстанция на обекта. Конкретно в Spring е разработен интерфейс, който играе ролята на контейнера за инжектиране на зависимости, а той е именно *org.springframework.beans.factory.BeanFactory* [20].

В Java идеалният клас е този, който е самостоятелен и не зависи от други класове. По този начин, функционалността на всеки отделен обект може да бъде разширявана и променяна без това да повлияе на останалите. Dependency Injection процесите, осигуряват именно това.

В текущия проект е използван Spring Boot. Този проект е на компанията Pivotal Software е направен с цел разработката на приложения и услуги, с минимални усилия и време от страна на програмиста. Може да се използва за създаване на самостоятелни Java приложения, които се пакетират до JAR файл, който лесно може да бъде стартиран с командата „java -jar“ [21]. В допълнение Spring Boot конфигурира всичко възможно автоматично. Заедно с това включва и функции, от който един софтуер се нуждае при пускането в експлоатация. Автоматично се създават крайни точки (или на английски endpoints), които могат да бъдат използвани за следене състоянието на приложението, предоставяйки основни показатели [21]. Следващото изображение показва, как Spring Boot позволява на потребителите да извлекат максимално от Spring платформата.



***Фиг 2.1 Spring Boot simple diagram***

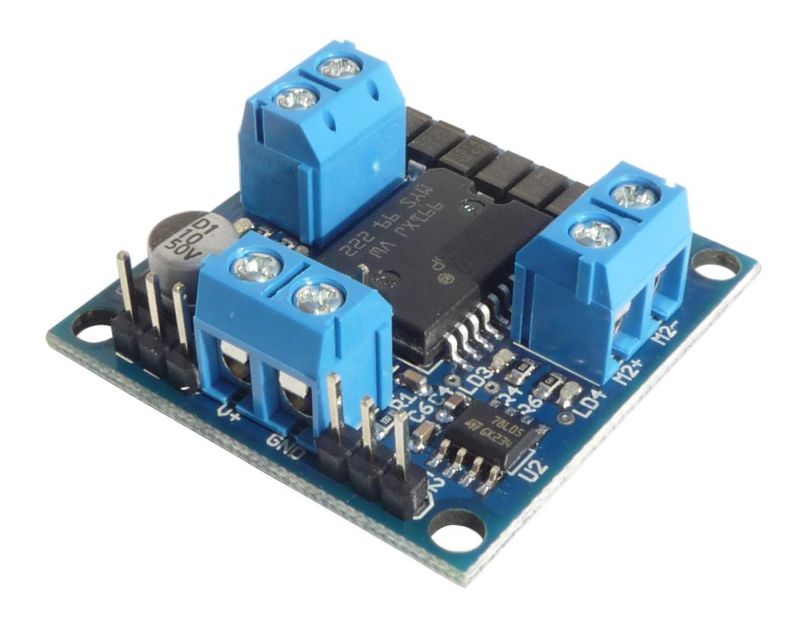
Диаграмата на Фиг 2.1е използвана от [21]. Естествено Spring Boot има някои съществени разлики, с които се отличава от Spring платформата. Например, основна разлика е че не генерира код и няма абсолютно никакво изискване за конфигуриране на XML [21]. Spring Boot поддържа Apache Tomcat (по подразбиране), Jetty и Undertow сървъри, защото те са съвместими с виртуалната машина на Java. За конкретния проект е използван сървърът по подразбиране – Apache Tomcat.

2.8 Използвани електронни елементи

В точка 2.5 от документацията е описан основният развоен хардуер – едночиповата компютърна система Raspberry Pi 3. Но, в проекта се налага използването на други електронни елементи. В тази част от втора глава ще бъдат разгледани два хардуерни компонента – Microbot H-bridge Motor Driver и Raspberry Pi камера модул.

2.8.1 Microbot DC Dual H-bridge Motor Driver

В текущия проект за управлението на моторите на колата е използван Н-мостов драйвер. Той е изработен от фирмата Microbot и осигурява независимо движение на два мотора управлявайки, както скоростта, така и посоката на въртене. Базиран е на известната интегрална схема L298, която е производство на фирмата STMicroelectronics. Проектиран е на входа да приема стандартни логически нива, а на изхода да управлява релета, мотори и други индуктивни товари. Минималното допустимо захранващо напрежение на този модул е 7 V. Затова като захранващ източник може да се използват литиево полимерни (LiPo) батерии, тъй като се характеризират сравнително малки размери и ниско тегло. Максималното допустимо захранващо напрежение на модула е 30 V.

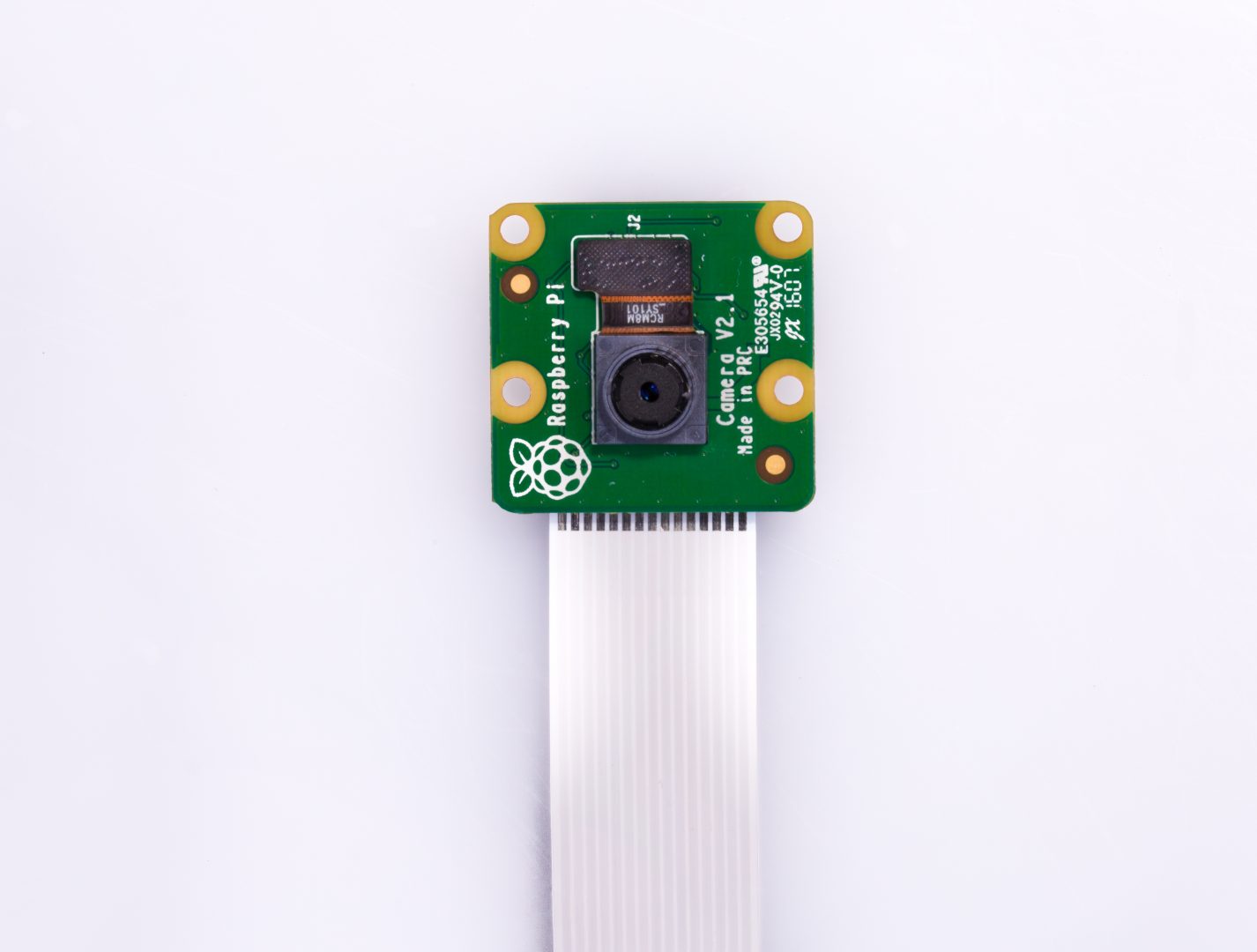


***Фиг 2.2 Microbot DC Dual H-Bridge Motor Driver***

Изображението показано на Фиг 2.2 принадлежи на [22]. Изходният ток е сравнително голям за този тип драйвери – около 4 A (по 2 А за всеки изходен канал).

2.8.2 Raspberry Pi Camera Module

Raspberry Pi Camera Module е специфичен модул, проектиран като вид разширение към основния хардуер на Raspberry Pi.



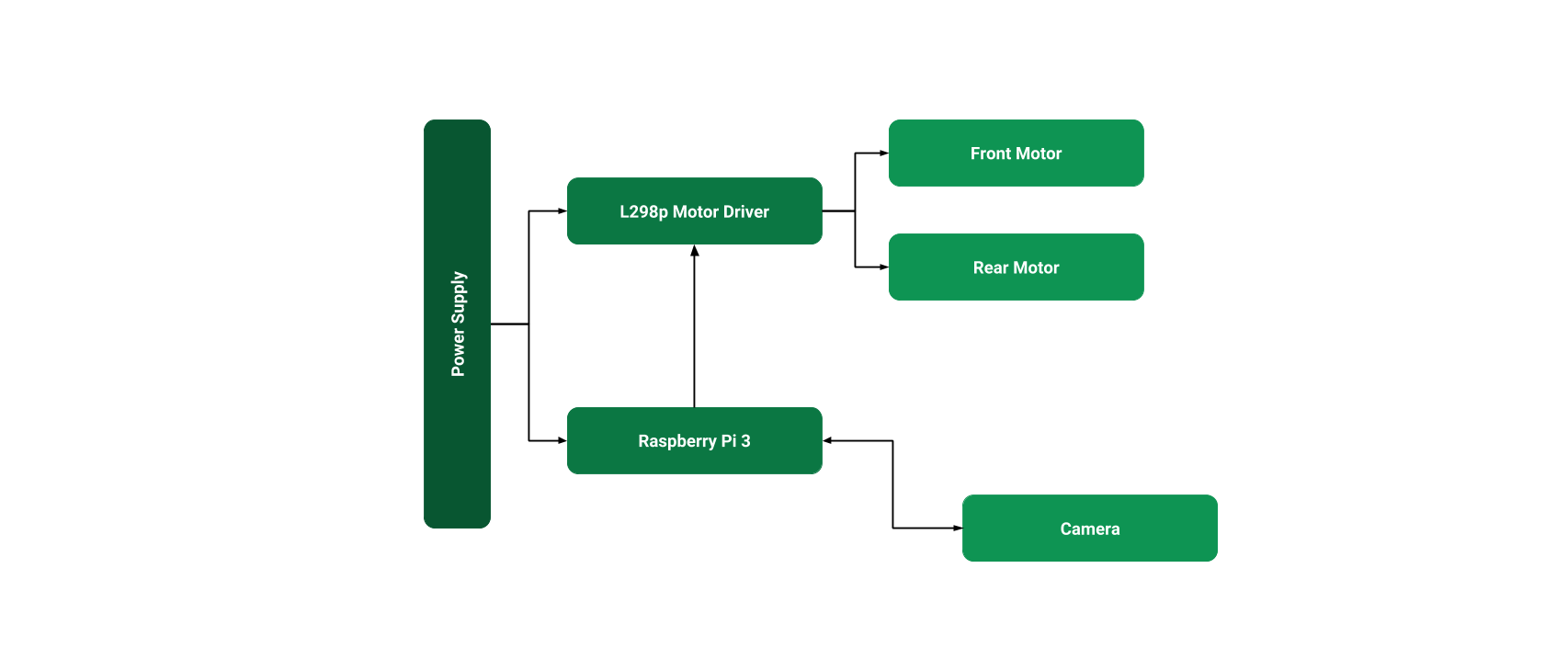
***Фиг 2.3 Camera Module v2.1***

На изображението, което е взето от [23], се вижда камера-модулът отпред. Свързва се към специфичен слот намиращ се върху корпуса на Raspberry Pi чипа, между HDMI изхода и 3.5 мм аудио жак. Този слот е специално създаден за свързване единствено на камери, защото комуникира през Camera Serial Interface (CSI).

CSI е интерфейс предназначен за предаване на единични изображения или видео между камера и процесор. Информацията по този интерфейс се предава на изключително високи скорости. Модулът притежава 8-мегапикселов сензор Sony IMX219 [23]. Поддържа максимална резолюция при снимки 3280 х 2464 пиксела, а при видео качеството е 1080p30, 720p60 and 640 × 480p60/90. Физически модулът се свързва към Pi чрез лентов кабел. По този начин камерата се свързва директно към централния процесор, посредством CSI шина. Шината е надеждна и е предназначена за пренасяне на пиксели от камерата към процесора. Естествено камерата се поддържа от последната операционна система създадена от Raspberry Pi Foundation – Raspbian. Камерата работи с всички компютърни системи Raspberry Pi. За достъпване на самата камера са разработени множество приложения и библиотеки. По-широкоразпространена библиотека за камерата е написана на Python и се нарича Picamera Python [23]. В крайна сметка този модул става много популярен и започва да се използва във все повече проекти.

ТРЕТА ГЛАВА

ПРОЕКТИРАНЕ И РЕАЛИЗАЦИЯ НА РОБОТИЗИРАНАТА КОЛА И ПРИЛОЖЕНИЕТО ЗА УПРАВЛЕНИЕ

3.1 Блокова схема на свързване на хардуерните елементи

***Фиг 3.1 Блокова схема на свързване***

На Фиг 3.1 се вижда принципна блокова схема, която показва как са свързани хардуерните елементи. Всеки блок представлява някакъв електронен елемент или устройство. По-подробно елементите ще бъдат разгледани в следващата глава. Блоковете ще се разглеждат от ляво надясно. Именувани са много разбираемо и не се нуждаят от много пояснения.

- Power Supply – има за цел да осигури захранване на драйвера, който ще управлява моторите, както и на едночиповата компютърна система Raspberry Pi 3.

- Raspberry Pi 3 - това е „ядрото“ на целия проект. Този блок играе ролята на контролен блок, тъй като управлява камерата и драйвера за моторите, през различни серийни интерфейси.

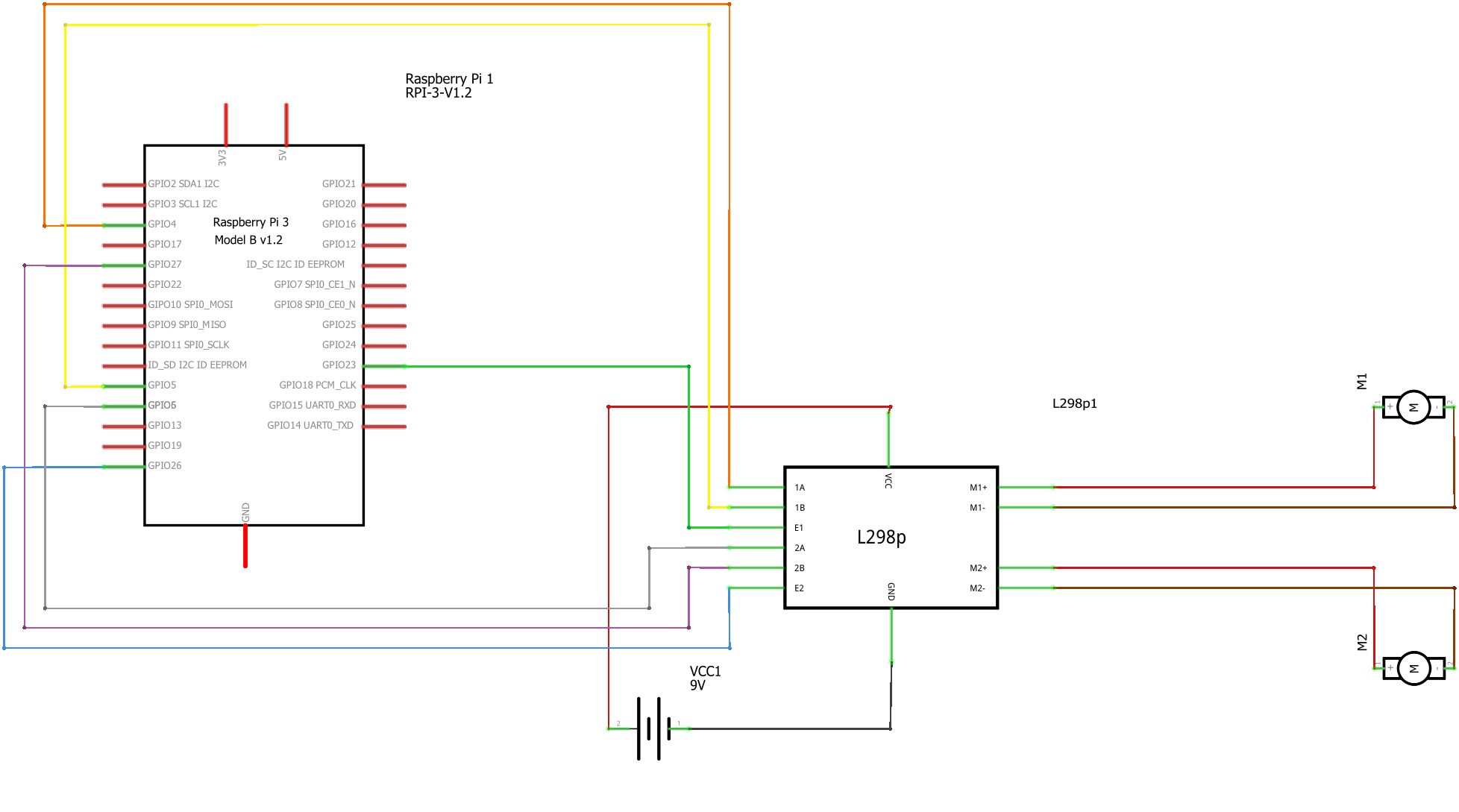
- Camera - представлява камерата, която ще бъде поставяна върху корпуса на колата. Тя е свързана само с контролния блок – Raspberry Pi 3.

- L298p Motor Driver - както името му подсказва този блок представлява драйверът, който управлява независимо двата мотора, задвижващи колата. Драйверът се нуждае от допълнително захранване, затова е свързан директно към захранващия блок. Но, за да изпълнява функцията си коректно и да има кой да го управлява е свързан и към Raspberry Pi блока.

- Front и Rear Motor – представляват двата мотора, които притежава колата, използвана за този проект. Front Motor – е предният мотор, той отговаря за завиването наляво и надясно, а Rear Motor – отговаря за движението на колата напред и назад. Естествено тези два блока най-често се използват комбинирано, като по този начин се осигурява движението в следните посоки: напред и надясно, напред и наляво, назад и надясно, назад и наляво. Свързани са само с драйвера, тъй като той е отговорен за тяхното управление.

Тази принципна блокова диаграма е неизменна част от проектирането и конструирането на една вградена система. В настоящия проект диаграмата е сравнително проста, но съществуват много по-големи проекти с множество отделни блокове. Благодарение на този тип диаграми човек има възможност лесно да се ориентира как са свързани и обособени дадени хардуерни компоненти.

3.2 Електрическа схема



***Фиг 3.2 Електрическа схема на свързване***

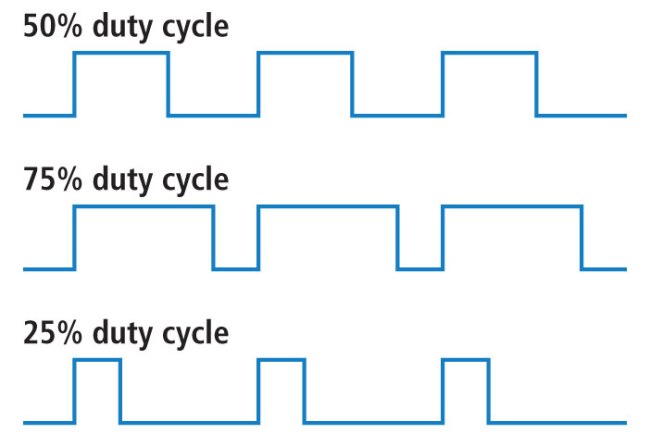
На Фиг 3.2 е изобразена електрическа схема на свързване на основните електронни елементи. Всеки компонент, участващ в схемата си има съответното означение:

* RPI-3-V1.2 – едночиповата компютърна система Raspberry Pi 3. Tя е основният елемент в схемата, осигуряваща управлението на моторите и възпроизвеждането на видео, посредством камера модул. Системата се захранва с напрежение от 5 V. Шест входно-изходни пина са свързани към драйвера за моторите. Всеки от тези пинове генерира напрежение от 3.3 V, при високо ниво и 0 V при ниско ниво.
* L298p – това означението на драйвера, който отговаря за управлението на моторите. Способен е да управлява два мотора независимо. Този чип може да бъде захранван с широк диапазон от входни напрежения от 7 – 30 V, конкретно в схемата е захранван от 9 V батерия. Има шест пина, чрез който се управляват моторите – E1, 1A, 1B, E2, 2A и 2B. Пиновете, който са означени с A и В, управляват посоката на движение на даден мотор. E1 и E2 (на английски: Enable Pin) показват, кой от двата мотора се управлява в даден момент, когато на E1 се подаде единица, тогава моторът, който се управлява е първият.
* M1 и M2 – така са означени двата мотора. В проекта M1 е задният мотор, отговарящ за движението на колата напред и назад. М2 e предният мотор, който определя посоката на завиване, дали наляво или надясно. Всеки от тези мотори притежават два извода + и -. Управлението им се извършва посредством широчинно-импулсна модулация на сигнала или на английски Pulse Width Modulation (PWM). Като в зависимост от големината на напрежението, което се подава върху + на мотора, скоростта на движение се променя. Колкото по-голямо напрежение се подаде, толкова по-бързо ще се върти даден мотор.

3.3 Широчинно-импулсна модулация (ШИМ)

Широчинно-импулсната модулация е процес, чрез който се управлява напрежението, следователно и мощността, подавана към даден товар. В този проект, тази техника се използва за контролирането на двата мотора. В основата на този механизъм стой един ключ, между захранването и товара, който променя състоянието си при голяма честота. Колкото по-голям е периодът, в който е включен, толкова е по-голяма е мощността подавана към товара. Основен термин при този процес е коефициентът на запълване (на английски: duty cycle). В този случай, той показва отношението на периода, през които ключът е в състояние „включен“, към периода през който е „изключен“. Може да има стойности от 0 до 1, където 0 е 0%, а 1 е 100%. На практика, като стойността му е 0, през целия период ключът е в изключено състояние. Едно много важно предимство на този тип модулация на сигнал, е че загубите на мощност са незначителни.

***Фиг 3.3 Графика показваща широчинно-импулсната модулация***



3.4 Използвани анотации на Spring в проекта

Spring Boot използва анотации, вместо стандартната XML конфигурация на приложенията. В тази част от документацията ще бъдат разгледани анотациите на Spring, който са използвани за реализацията на проекта.

*@SpringBootApplication –* това е основна анотация, като нейната функционалност е задължителна за всеки проект използващ Spring Boot. Използвана е в основния клас на сървърната част. Тя е създадена на мястото на три други анотации, които са широко използвани - *@Configuration, @EnableAutoConfiguration* и *@ComponentScan*. Основното нещо, което прави тази анотация е да сканира за класове, които са компоненти на Spring. Особеното е, че тя сканира само пакета в който се намира и под-пакетите му.

*@Component* – това е друга анотация, която е използвана в сървърната част на проекта. Тя маркира даден Java клас като основен за структурата на приложението, така че механизмът за сканиране на компоненти на Spring да добави съответния клас към контекста на приложението.

*@RestController –* тази анотация маркира даден клас, като контролер, където методите не е задължително да връщат *@ResponseBody*. Това е полезно, защото не се налага да връщаш HTML например, а може да се върне например JSON форматиран отговор.

@RequestMapping – в конкретиката на приложението, тази анотация е използвана на ниво метод. Когато се използва върху даден метод, тя предоставя идентификатор (URI), на който ще бъдат извикани методите за обработка. Пример за използване на тази анотация в кода е:

@RequestMapping(method = RequestMethod.***GET***,  
 value = **"/direction/forwardOrBackward"**,  
 produces = MediaType.***TEXT\_PLAIN\_VALUE***)  
**public** Integer getSpeedValue() { … }

***Фиг 3.4 Пример за използването на @RequestMapping анотацията***

3.5 Сървърна част на проекта

Комуникацията между развойната едночипова система на колата и мобилния телефон е безжична през WiFi. Ocъществява се чрез най-разпространеният протокол за комуникация по мрежата - HTTP. Приложението спазва архитектурният модел - Representational State Transfer (REST). Именно защото придържането към този модел, изисква спазването на няколко ограничения, сървърната и клиентската част разделени. Сървърната част на проекта представлява уеб приложение, написано на Java. Приложението използва Spring Boot проекта, за който беше разказано във втора глава. Основната задача, която трябва да изпълнява приложението е да пакетира Tomcat сървър и да създава самостоятелен файл, с разширение .jar, който да бъде качен на Raspberry Pi и да бъде лесно стартиран с командата: *sudo java -jar <NameOfApplicationSnapshot>.jar*. За да може приложението да бъде стартирано по начина, описан в предишните изречения, Spring създава специален клас. В проекта този клас се нарича *org.elsys.remote\_control\_car.RemoteControlCarApplication*.

@SpringBootApplication  
@ComponentScan(basePackages = **org.elsys.remote\_control\_car.controller"**)  
**public class** RemoteControlCarApplication {  
  
 **public static void** main(String[] args) {  
 SpringApplication.*run*(RemoteControlCarApplication.**class**, args);  
 }  
}

***Фиг 3.4 RemoteControlCarApplication класът***

3.5.1 IP конфигурация на Raspberry Pi

За да е възможно колата да бъде управлявана коректно, чрез смартфон, в проекта те са свързани към една мрежа. На Raspberry Pi е зададен статичен IP адрес - *169.254.148.138*. По този начин, когато сървърът стартира, върху Pi той винаги ще е на адрес *169.254.148.138:8080*, като 8080 е портът по подразбиране на Tomcat базиран сървър. Това естествено си има недостатъци, защото за някои мрежи, този адрес няма да може да бъде зададен и ще трябва да се премине към динамично конфигуриране на IP адрес (DHCP).

3.5.2 Библиотека за управление на входно-изходни операции върху Pi

За реализирането на управлението на колата, е използван проектът - Pi4J. Той предоставя възможност да се достъпят входно-изходните пинове с общо предназначение (на английски: GPIO), като се използва програмният език Java. Pi4J от своя страна имплементира Java Native Interface (JNI) обвивка на библиотеката WiringPi. Java Native Interface е механизъм при който виртуалната машина на Java, може да извиква „естествен“ (на английски: native) код, компилиран от езици от по-ниско ниво, като C. В допълнение, Pi4J няма нужда от допълнителни зависимости по време на компилация и изпълнение.

3.5.3 Управление на колата и крайни точки на приложението

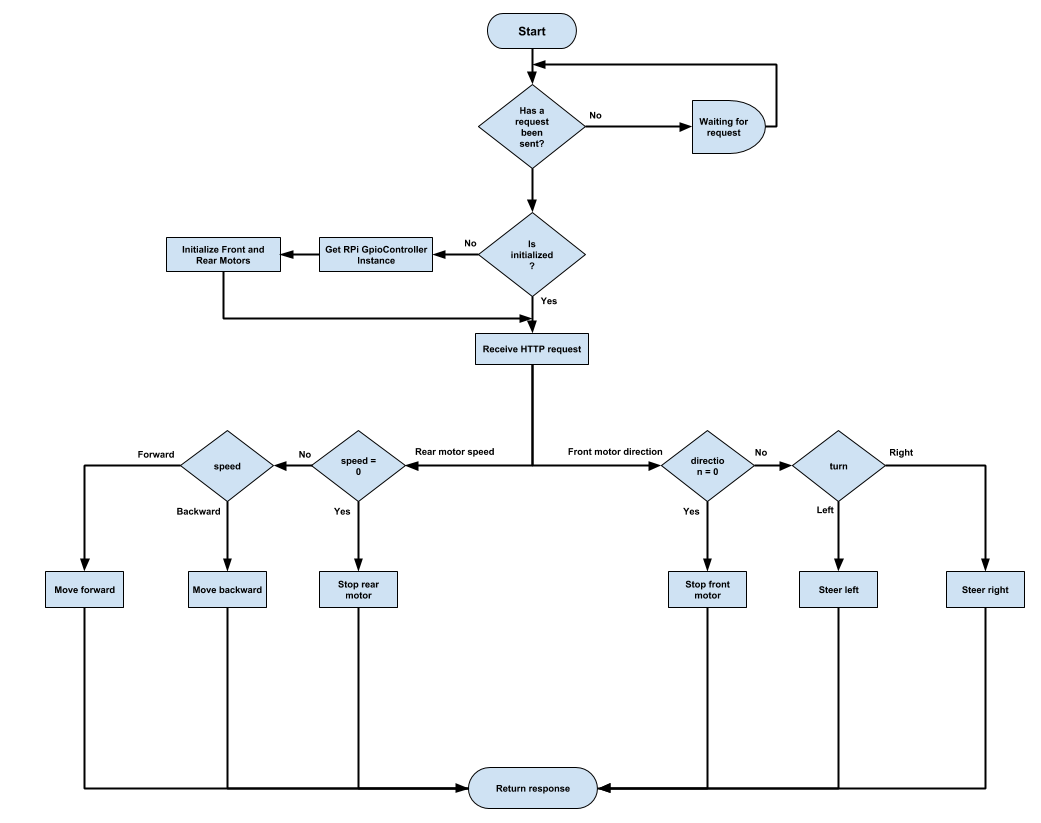
Приложението приема и обработва само GET и POST заявки базирани на HTTP протокола. Създаден е набор от крайни точки (на английски: endpoints), чрез който да се осъществи достъпа до Raspberry Pi и управлението на колата:

* ***POST*** *raspberrypi\_IP:8080/direction/forwardOrBackward/{speed} –* чрез тази заявка се задава скоростта *{speed}* на мотора за движение на колата напред или назад. Моторът се стартира със зададената скорост *{speed},* ако тя е различна от нула. Посоката на движение се определя в зависимост от знака на *{speed}*:
* Ако е отрицателно число (< 0) се задава посока на движение - назад.
* Ако е нула, значи към мотора не се подава напрежение и той е спрян.
* Ако е положително число (> 0) се задава посока на движение - напред.
* ***GET*** *raspberrypi\_IP:8080/direction/forwardOrBackward/ –* тази заявка връща като отговор число, показващо текущата скорост на мотора за движение напред или назад.
* ***POST*** *raspberrypi\_IP:8080/direction/leftOrRight/{turn} –* чрез тази заявка се задава посоката *{turn}* на завиване на мотора – наляво или надясно. Моторът се стартира със зададената посока *{turn},* ако тя е различна от нула. Посоката на завиване се определя в зависимост от стойността и знака на *{turn}*:
* Ако е отрицателно число (- 1) се задава посока на завиване – надясно.
* Ако е нула, значи към мотора не се подава напрежение и той е спрян.
* Ако е положително число (1) се задава посока на завиване – наляво.
* ***GET*** *raspberrypi\_IP:8080/direction/leftOrRight/ –* тази заявка връща като отговор число, показващо текущото положение на мотора за завиване наляво или надясно.

В два от идентификаторите има параметри {speed} и {turn}, които променят стойността си динамично и в случая имат много важна роля при управлението на колата. Затова в приложението е използвана анотацията @PathVariable, която се използва за деклариране на такъв тип параметри. Взимането на скоростта на мотора от адреса, като параметър става по следния начин:

@RequestMapping(method = RequestMethod.***POST***,  
 value = **"/direction/forwardOrBackward/{speed}"**)  
**public** ResponseEntity runForwardOrBackward(@PathVariable(**"speed"**) **int** speed) { ... }

***Фиг 3.5 Пример за използване на @PathVariable и @RequestMapping***

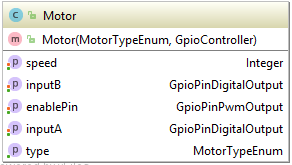
****

***Фиг 3.6 Диаграма, показваща алгоритъма на работа за движението на колата***

На Фиг 3.6 се вижда диаграма, която показва алгоритъма на работа при управлението на колата или с други думи какво се изпълнява когато някоя от гореописаните заявки бъде получена.

3.4.5 Инициализация на обектите от тип Motor

Класът описва характеристиките на моторите на колата. Диаграмата на самия клас може да бъде видяна на следващата фигура.



***Фиг 3.7 Класът Motor***

Класът притежава един конструктор, в който се инициализират моторите – техния тип, входно-изходните пинове за управление, както и скоростта на движение. Инициализацията на моторите е реализирана със следния фрагмент от код:

**public** Motor(MotorTypeEnum type, GpioController gpioController) {  
 **this**.**type** = type;  
  
 **switch** (**this**.**type**) {  
 **case *REAR***:  
 **this**.**inputA** = gpioController.provisionDigitalOutputPin(RaspiPin.***GPIO\_04***, **"1A"**, PinState.***LOW***);  
 **this**.**inputB** = gpioController.provisionDigitalOutputPin(RaspiPin.***GPIO\_05***, **"1B"**, PinState.***LOW***);  
 **this**.**enablePin** = gpioController.provisionPwmOutputPin(RaspiPin.***GPIO\_23***, **"E1"**, 0);  
 **break**;  
 **case *FRONT***:  
 **this**.**inputA** = gpioController.provisionDigitalOutputPin(RaspiPin.***GPIO\_06***, **"2A"**, PinState.***LOW***);  
 **this**.**inputB** = gpioController.provisionDigitalOutputPin(RaspiPin.***GPIO\_27***, **"2B"**, PinState.***LOW***);  
 **this**.**enablePin** = gpioController.provisionPwmOutputPin(RaspiPin.***GPIO\_26***, **"E2"**, 0);  
 **break**;  
 }  
  
 **this**.**speed** = 0;  
}

***Фиг 3.8 Конструкторът на класа Motor***

Първият аргумент на конструктора е MotorTypeEnum – това е изброим тип (на английски: Enumeration). Има две стойности REAR и FRONT, което в случая представлява типът на двата мотора – преден и заден.

**public enum** MotorTypeEnum {  
 ***FRONT***,  
 ***REAR*** }

***Фиг 3.9 Изброимият тип MotorTypeEnum***

Вторият аргумент на конструктора е от тип – GpioController, интерфейс част от Pi4J проекта. Интерфейсът описва на практика всички операции върху входно-изходните пинове, с общо предназначение. В кода са използвани два метода за същинското инициализиране на моторите. Те са:

* GpioPinDigitalOutput **provisionDigitalOutputPin**(Pin pin, String name, PinState defaultState);
* GpioPinPwmOutput **provisionPwmOutputPin**(Pin pin, String name, **int** defaultValue);

След като разгледахме дефиницията на конструктора, следват свойствата (на английски properties):

* **private** Integer **speed**; - това свойство представлява скоростта на мотора.

Тъй като се използва процесът на широчинно-импулсна модулация, за контролиране скоростта на движение или степента на завиване, стойността на този параметър на мотора, може да бъде в диапазона от 0 – 1024.

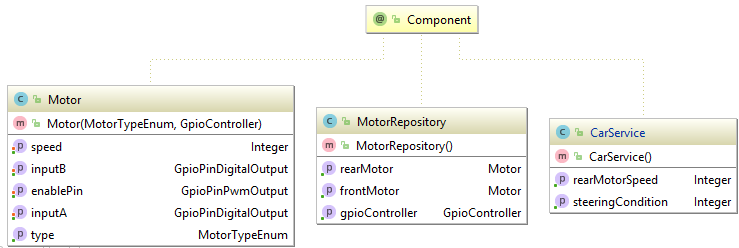
* **private** GpioPinDigitalOutput **inputA**, **inputB**; - чрез тези две свойства, се управлява посоката на движение/завиване на мотора. Техните стойности могат да са или 0, или 1. Но, в кода е използван изброимият тип PinState, предоставен от Pi4J проекта - PinState.LOW е 0, а PinState.HIGH e 1.
* **private** GpioPinPwmOutput **enablePin**; - този пин се използва за контролиране на скоростта на движение на съответния мотор. Стойността се задава в следния метод:

**public void** setSpeed(Integer speed) {  
 **this**.**speed** = speed;  
 getEnablePin().setPwm(**this**.**speed**);  
 }

***Фиг 3.10 Методът Motor::setSpeed(Integer speed)***

Този клас беше подробно описан, защото е основен за движението на колата и показва взаимодействието между Java и входно-изходните пинове на Raspberry Pi.

3.4.6 Структура на класовете в приложението

****

***Фиг 3.11 Класовете, които са компоненти на Spring***

В предишната точка 3.4.5 беше описан начина на инициализация на моторите, както и самия Motor клас. В приложението са използвани още два класа, които притежавайки анотация @Component, биват добавяни към контекста на Spring приложението.

3.4.6.1 Класът MotorRepository

Този клас изпълнява основната функционалност на приложението. Съдържа в себе си едно поле от типа GpioController, което е статично защото ни е нужна само една инстанция от този обект, за да манипулираме входно-изходните пинове на развойния хардуер.

**private static** GpioController *gpioController* = GpioFactory.*getInstance*();

***Фиг 3.12 Инстанциране на обект от тип GpioController***

Управлението на моторите е имплементирано в 4 метода принадлежащи на класа MotorRepository. Като пример ще вземем реализирането на движението на мотора напред/назад:

**public void** runRearMotor(Integer speed, DirectionEnum direction) {  
  
 *run*(direction, **this**.**rearMotor**.getInputA(), **this**.**rearMotor**.getInputB());  
  
 **this**.**rearMotor**.setSpeed(speed);  
}  
  
**private void** run(DirectionEnum direction, GpioPinDigitalOutput inputA, GpioPinDigitalOutput inputB) {  
 **switch** (direction) {  
 **case *FORWARD***:  
 inputA.setState(PinState.***LOW***);  
 inputB.setState(PinState.***HIGH***);  
 **break**;  
 **case *BACKWARD***:  
 inputA.setState(PinState.***HIGH***);  
 inputB.setState(PinState.***LOW***);  
 **break**;  
 **case *NONE***:  
 inputA.setState(PinState.***LOW***);  
 inputB.setState(PinState.***LOW***);  
 **break**;  
 }  
}

***Фиг 3.13 Методите MotorRepository::runRearMotor и MotorRepository::run***

3.4.6.2 Класът CarService

На Фиг 3.6 е представена диаграмата, която показва как е реализиран алгоритъмът за управление на колата. Има няколко проверки, които трябва да се направят преди даден мотор, да бъде задвижен в някаква посока. Тези проверки, са направени в класа CarService. Този клас съдържа в себе си инстанция на MotorRepository, като от нея извиква методите за контрол на моторите. Отново примерът е със задвижването на колата напред или назад.

**public void** runMotorForwardOrBackward(Integer speed) {  
 **if** (speed != 0) {  
 **if** (speed > 0) {  
 **motorRepository**.runRearMotor(speed, DirectionEnum.***FORWARD***);  
 **return**;  
 }  
  
 **motorRepository**.runRearMotor(speed, DirectionEnum.***BACKWARD***);  
 **return**;  
 }  
  
 **motorRepository**.runRearMotor(speed, DirectionEnum.***NONE***);  
}

***Фиг 3.14 Методът CarService::runMotorForwardOrBackward***

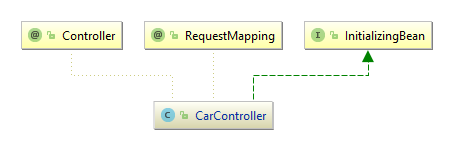
Тук е момента да се обърне внимание, че за по-добър стил на писане, е използван изброим тип, който да дефинира посоките на движение. Това е DirectionEnum.

**public enum** DirectionEnum {  
 ***FORWARD***,  
 ***BACKWARD***,  
 ***LEFT***,  
 ***RIGHT***,  
 ***NONE***}

***Фиг 3.15 Изброимият тип DirectionEnum***

3.4.6.3 Класът CarController

Следващата фигура представлява диаграма на класа CarController.



***Фиг 3.16 Класът CarController и Spring дефинирани анотации***

Основната задача на контролера е да приема входящите заявки и спрямо тях да извиква класове и методи, които да ги обработят. В приложението, контролерът е класът CarController. Маркиран е с анотация @RestController, се указва на Spring, че това е контролер. Особеното в случая е, че този клас имплементира специален конфигурационен интерфейс InitializingBean. Нуждата от предефинирането на този интерфейс е, защото обекта от тип CarService трябва да бъде инициализиран след като се конфигурира приложението. Това става с предефинирането на метода afterPropertiesSet().

@RestController  
@RequestMapping(**"/api/v1"**)  
**public class** CarController **implements** InitializingBean {  
  
 **private** CarService **carService**;  
  
 @Override  
 **public void** afterPropertiesSet() **throws** Exception {  
 **this**.**carService** = **new** CarService();  
 }  
}

***Фиг 3.17 Предефиниране на метода afterPropertiesSet() в CarController класа***

Дотук беше описана сървърната част от проекта. В следващата част от трета глава, ще бъде разгледано Android приложението.

3.6 Клиентска част на проекта

В тази точка ще бъде описана клиентската част на проекта. Тя представлява мобилно приложение, базирано на Android операционна система. Езикът използван за неговата реализация е Java. Предназначението на приложението е да функционира, като дистанционно управление. Чрез натискате на бутони, се изпращат HTTP заявки, които се обработват от сървъра и контролират посоката на движение на колата.

3.6.1 Реализиране на шаблони за дизайн

Шаблоните за дизайн (на английски: Design Patterns) представляват универсални решения на често срещани проблеми, по време на разработка.

В това Android приложение, някой класове имплементират, два от най-популярните шаблони - Singleton и Factory Method. В тази точка ще бъдат разгледани накратко характеристиките на всеки от тях.

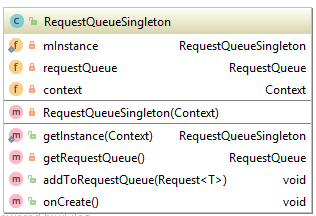
* Singleton – това може би е най-широкоразпространеният шаблон. Характеризира се с това, че позволява на даден клас, да има само една инстанция. Класът от своя страна, притежава метод, който връща тази инстанция. За да се предотврати създаването на други обекти конструкторът на класа, имплементиращ този шаблон е частен (на английски: private).
* Factory Method – това е друг шаблон, който е използван в реализацията на приложението. Както името му подсказва, клас който имплементира този шаблон трябва да създава („произвежда“) обекти. Инстанцирането на обектите се осъществява през специален метод. В зависимост от някакъв външен параметър, методът решава кой обект да създаде.

3.6.2 Библиотека за изпращане на HTTP заявки

За реализирането на механизма за изпращане на HTTP заявки от Android приложението е използвана една от библиотеките на Google, наречена Volley. Тя предоставя много лесен начин за генериране на заявки и тяхното изпращане, без да се използват пряко класове като *java.net.HttpURLConnection*.

3.6.3 Класът RequestQueueSingleton

Библиотеката Google Volley, която беше разгледана в предишната точка, използва специален клас за транспортиране на заявките и обработка на кеширането. Този клас се нарича RequestQueue. Тъй като се налага постоянното изпращане на HTTP заявки от страна на приложението най-ефективно ще е инстанцията на RequestQueue да е само една. Ето защо за реализацията на класът RequestQueueSingleton е използван шаблонът Singleton (описан в точка 3.6.1).



***Фиг 3.18 Класът RequestQueueSingleton***

Инстанцията на класа се взима от метода getInstance(Context), той е реазлизиран в следния фрагмент от код [25]:

**public static synchronized** RequestQueueSingleton getInstance(Context context) {

**if** (*mInstance* == **null**) {  
 *mInstance* = **new** RequestQueueSingleton(context);  
 }  
 **return** *mInstance*;  
}

***Фиг 3.19 Медотът getInstance(Context)***

Тъй като инстанцията на този клас може да бъде използвана от повече от една нишка, затова методът е synchronized. По този начин се избягва момента, когато две нишки могат да достъпят обекта едновременно. Това е механизъм, който предотвратява възникването на грешки при управлението на паметта. Параметърът от тип Context предоставя информация за средата на приложението. За добавяне на заявки към опашката се грижи следният метод [25]:

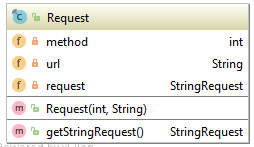
**public** <T> **void** addToRequestQueue(Request<T> req) {  
 getRequestQueue().add(req);  
}

***Фиг 3.20 Медотът addToRequestQueue(Request<T> req)***

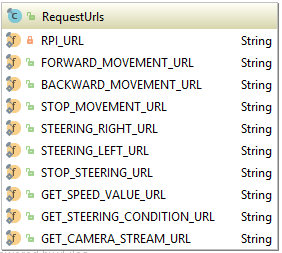
След като заявката бъде добавена към RequestQueue тя бива изпращана към сървъра.

3.6.4 Класът org.elsys.remote\_control\_car.request.Request

Класът описва заяката, която се изпраща към сървъра за управлението на колата.



***Фиг 3.21 Класът org.elsys.remote\_control\_car.request.Request***

Инстанцирането на обектите от тип Request се реализира с два параметъра, методът на HTTP заявката, като в проекта са използвани два типа заявки GET и POST и символен низ, който представлява адреса, към който се изпраща заявката. Адресите, чрез който се достъпва сървъра и неговите ресурси се пазят в отделен клас, наречен RequestUrls. Реализирани са като константи от тип String.

***Фиг 3.22 Класът RequestUrls***

Както беше отбелязано в точка 3.5.1 за момента IP адресът на Raspberry Pi е статичен. Ето защо стойността на RPI\_URL е следната и за пример ще вземем адресът, при който колата се задвижва напред FORWARD\_MOVEMENT\_URL.

**private static final** String ***RPI\_URL*** = **"http://169.254.148.138:8080"**;  
  
 **public static final** String ***FORWARD\_MOVEMENT\_URL*** = ***RPI\_URL*** + **"/direction/forwardOrBackward/200"**;

***Фиг 3.23 Стойностите на RPI\_URL и FORWARD\_MOVEMENT\_URL***

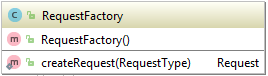
## Методът getStringRequest() на класа Request връща като резултат инстанция на com.android.volley.toolbox.StringRequest. Тази инстанция се добавя към RequestQueue и се изпраща към сървъра.

**public** StringRequest getStringRequest() {  
 **return new** StringRequest(  
 **this**.**method**,  
 **this**.**url**,  
 response -> Log.*d*(**"ON-SUCCESS"**, response),  
 error -> Log.*d*(**"ON-ERROR"**, error.getMessage())  
 );  
 }

***Фиг 3.24 Методът getStringRequest()***

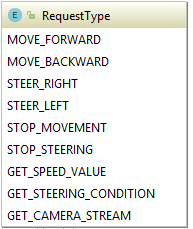
3.6.5 Класът RequestFactory

След като вече беше описан класът Request, трябва по някакъв начин различните обекти (заявки) да бъдат създавани. Ето защо е реализиран клас RequestFactory, който използва шаблонът за дизайн Factory Method описан в точка 3.6.1.



***Фиг 3.25 Класът RequestFactory***

На диаграмата се вижда че, класът има конструктор по подразбиране и един метод, който е статичен. Този метод има един параметър от изброимия тип RequestType. Чрез този единствен параметър методът решава, каква инстанция на обект от тип Request да създаде. RequestType пази в себе си типовете на заявките, който могат да бъдат създадени.



***Фиг 3.26 Изброимият тип RequestType***

Следващата фигура е фрагмент от код, който показва как се създават обектите от тип Request в зависимост от типа на заявката.

**public static** Request createRequest(RequestType type) {  
 Request request = **null**;  
   
 **switch** (type) {

**case *MOVE\_FORWARD***:  
 request = **new** Request(com.android.volley.Request.Method.***POST***, RequestUrls.***FORWARD\_MOVEMENT\_URL***);  
 **case *MOVE\_BACKWARD***:  
 request = **new** Request(com.android.volley.Request.Method.***POST***, RequestUrls.***BACKWARD\_MOVEMENT\_URL***);  
 **case *STEER\_RIGHT***:  
 request = **new** Request(com.android.volley.Request.Method.***POST***, RequestUrls.***STEERING\_RIGHT\_URL***);  
 **case *STEER\_LEFT***:  
 request = **new** Request(com.android.volley.Request.Method.***POST***, RequestUrls.***STEERING\_LEFT\_URL***);  
 **case *STOP\_MOVEMENT***:  
 request = **new** Request(com.android.volley.Request.Method.***POST***, RequestUrls.***STOP\_MOVEMENT\_URL***);  
 **case *STOP\_STEERING***:  
 request = **new** Request(com.android.volley.Request.Method.***POST***, RequestUrls.***STOP\_STEERING\_URL***);  
 **case *GET\_SPEED\_VALUE***:  
 request = **new** Request(com.android.volley.Request.Method.***GET***, RequestUrls.***GET\_SPEED\_VALUE\_URL***);  
 **case *GET\_STEERING\_CONDITION***:  
 request = **new** Request(com.android.volley.Request.Method.***GET***, RequestUrls.***GET\_STEERING\_CONDITION\_URL***);

}

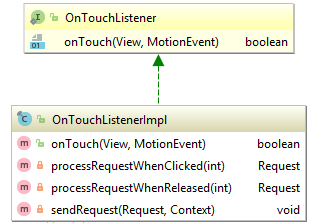
**return** request;  
}

***Фиг 3.27 Методът RequestFactory::createRequest(RequestType)***

3.6.6 Класът OnTouchListenerImpl

Тъй като основният екран на приложението се състой от няколко бутона, с който се управлява колата е ненадеждно за всеки от бутоните директно да имплементира View.OnTouchListener интерфейса на Android.

Вместо това най-удачно е един отделен клас да имплементира този интерфейс и целия механизъм да е в него. В текущото приложение този клас e OnTouchListenerImpl. Той имплементира метода onTouch(View, MotionEvent който се извиква когато някой от бутоните бъде натиснат.



***Фиг 3.28 Класът OnTouchListenerImpl***

Изпращането за заявка към сървъра става в метода sendRequest(Request, Context). Той е реализиран по следния начин:

**private void** sendRequest(org.elsys.remote\_control\_car.request.Request request, Context context) {

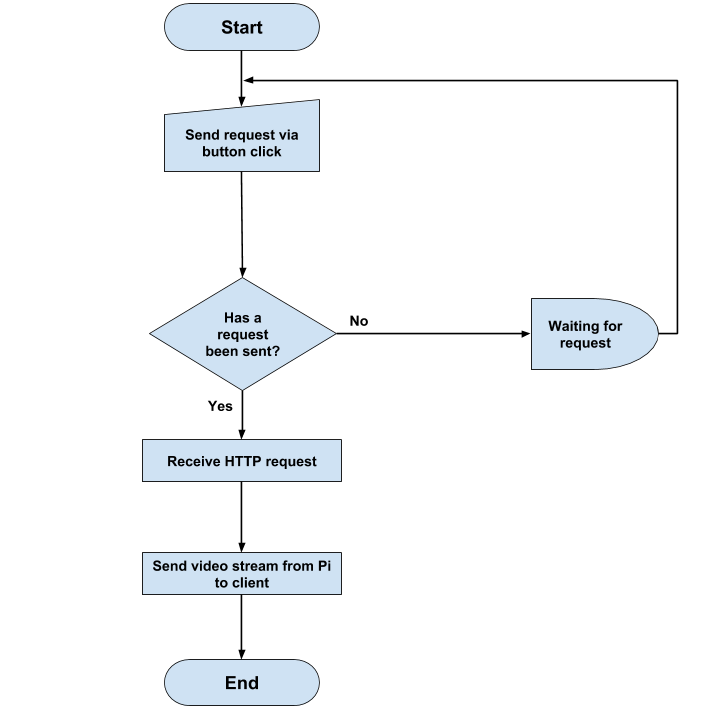
Request<String> stringRequest = request.getStringRequest();  
  
 RequestQueueSingleton.*getInstance*(context)  
 .addToRequestQueue(stringRequest);  
}

***Фиг 3.29 Методът sendRequest(Request, Context)***

3.7 Получаване на видео поток от камерата към Android приложението

Генерирането на видео потока става с помощта на инструмента от командния ред на Raspberry Pi – raspivid. Той предоставя набор от флагове (аргументи), който могат да модифицират изходния видео поток, както и да го пренасочват. За конкретния проект искаме видео потокът, излъчван от камерата да се изпраща от сървъра директно към Android приложенито затова използваме –output флага, като не уточняваме конкретен файл, в който да бъде записано видеото, а оставяме стойността на този параметър равна на -. По този начин видео потокът ще бъде излъчван през стандартния изходен поток (stdout). Така командата ще изглежда по следния начин: ***raspivid -o –.***

Следващата стъпка е да се добави време, през което да се заснема видеото – задава се с параметър --timeout, когато не е зададен по подразбиране е 5 секунди, но за проекта има стойност 0, тъй като искаме видео потокът да се изпраща, докато процесът не бъде терминиран. Командата вече изглежда така: ***raspivid -o - -t 0.*** Има още няколко основни параметъра на тази команда, които могат да бъдат зададени, като ширина и височина на видеото (--width, --height). Видео потокът се получава на следния адрес дефиниран в уеб приложението управляващо сървъра: ***GET*** *raspberrypi\_IP:8080/camera/stream.*



***Фиг 3.30 Диаграма, показваща процеса за получаване на видео от камерата***

ЧЕТВЪРТА ГЛАВА

РЪКОВОДСТВО НА ПОТРЕБИТЕЛЯ

4.1 Изисквания към компютърната конфигурация

Приложението се базира на Java платформата. Ето защо компютърната конфигурация задължително трябва да има инсталиран Java Runtime Environment (JRE). По този начин, самото приложение може да бъде стартирано върху всякакви операционни системи.

4.2 Необходими хардуерни елементи (Bill Of Materials)

За да сглоби колата потребителят, трябва да има следните електронни елементи:

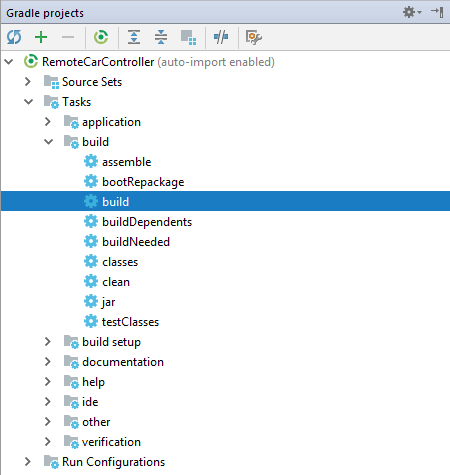
* Raspberry Pi 3 – като не е задължително да се използва точно този вид компютърна система, може да се използва подобна платка, като OrangePi или ASUS Tinker.
* Microbot DC Dual H-bridge L298p драйвер за моторите на колата – като отново не е задължително да се използва точно този модел драйвер.
* Raspberry Pi Camera Module – тъй като за реализацията на този проект е използвана Raspberry Pi 3, най-добре е да се използва модул, който е произведен специално за нея. Естествено може да се използват и други камери – например USB.
* SD карта – място, на което се инсталира операционната система на Raspberry Pi. По-добре картата да е възможно най-висок клас и с поне 16 GB памет.
* Батерия (тип Power Bank) – тя служи за захранване на хардуерна система

Други вторични хардуерни компоненти, който са нужни за връзването на елементите, са едножилни проводници - наречени джъмпери, както и поялник.

4.3 Инструкции за качване на уеб приложението на развойния хардуер

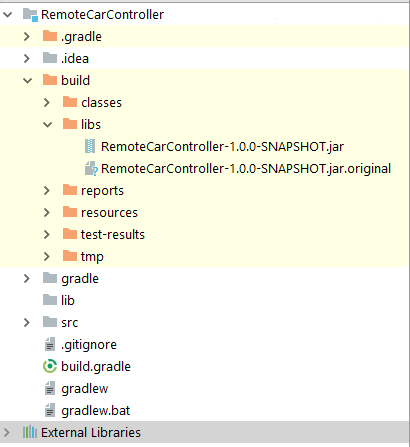
Връзката между Raspberry Pi и локалната машина, се осъществява като се свързват с RJ45 кабел. По този начин може да се активира SSH връзка.

**-** Първа стъпка при инсталиране на приложението на развойния хардуер, е то да бъде пакетирано в един jar файл. Това става посредством използването на инструментa Gradle, който се използва за изграждане на проекти. Всеки Gradle проект има дефинирани задачи (на английски: tasks). За да се пакетира проектът в jar файл, трябва да се изпълни „build“ задача.



***Фиг 4.1 Gradle „build“ task***

След като изпълнението на задачата приключи, във файловата структура на приложението се появява нова директория /build. В нея е генерираният jar.



***Фиг 4.2 Генериран jar файл на приложението***

**-** След като вече е генериран, файлът трябва да бъде качен на платката. Това се осъществява, по споменатата SSH връзка. За тази цел, може да се използва Linux командата **scp**. В конзолата (наречена в Linux „Terminal“) се пише следната команда**:**

*scp RemoteCarController-1.0.0-SNAPSHOT.jar pi@raspberryPi\_IP:/home/pi/*

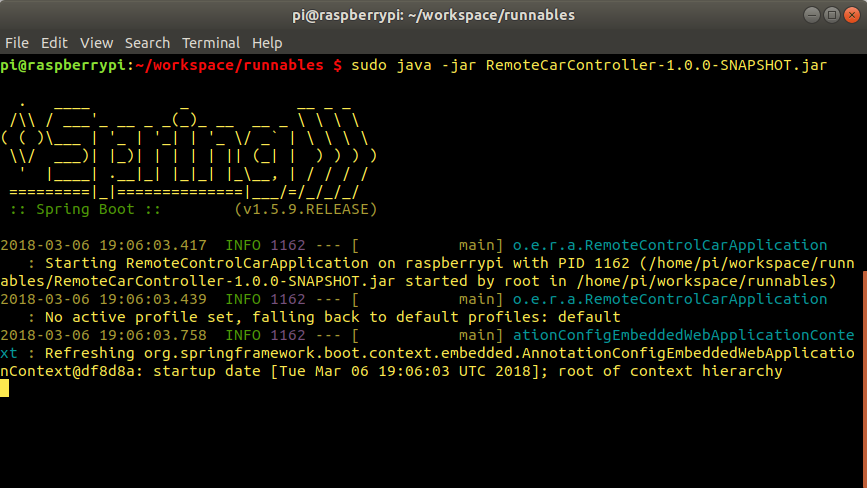
В зависимост от операционната система на локалната машина, има различни начини, по които може се осъществи SSH връзката. Ако е през Windows, трябва да се използва SSH клиент, като *Putty*. Ако е на Ubuntu се използва командата *ssh pi@raspberryPi\_IP*. За пример ще бъде използвана Ubuntu операционна система:

1. След въвеждане на *ssh pi@raspberryPi\_IP*, ще бъде поиската парола *ssh pi@raspberryPi\_IP’s password:*
2. По подразбиране паролата е **raspberry**
3. Ако паролата е въведена правилно, в конзолата ще се появи следният ред: **pi@raspberry ~ $,** това означава, че потребителят има достъп до платката

- Файлът вече е качен на Raspberry Pi и остава да бъде изпълнен. Последната стъпка е да се стартира уеб приложението, през командния ред със следната команда, която изпълнява .jar файлове:

**sudo java -jar RemoteCarController-1.0.0-SNAPSHOT.jar**

Стартиралото приложение, може да се види на следващата фигура:



***Фиг 4.3 Стартирало Spring Boot приложение***

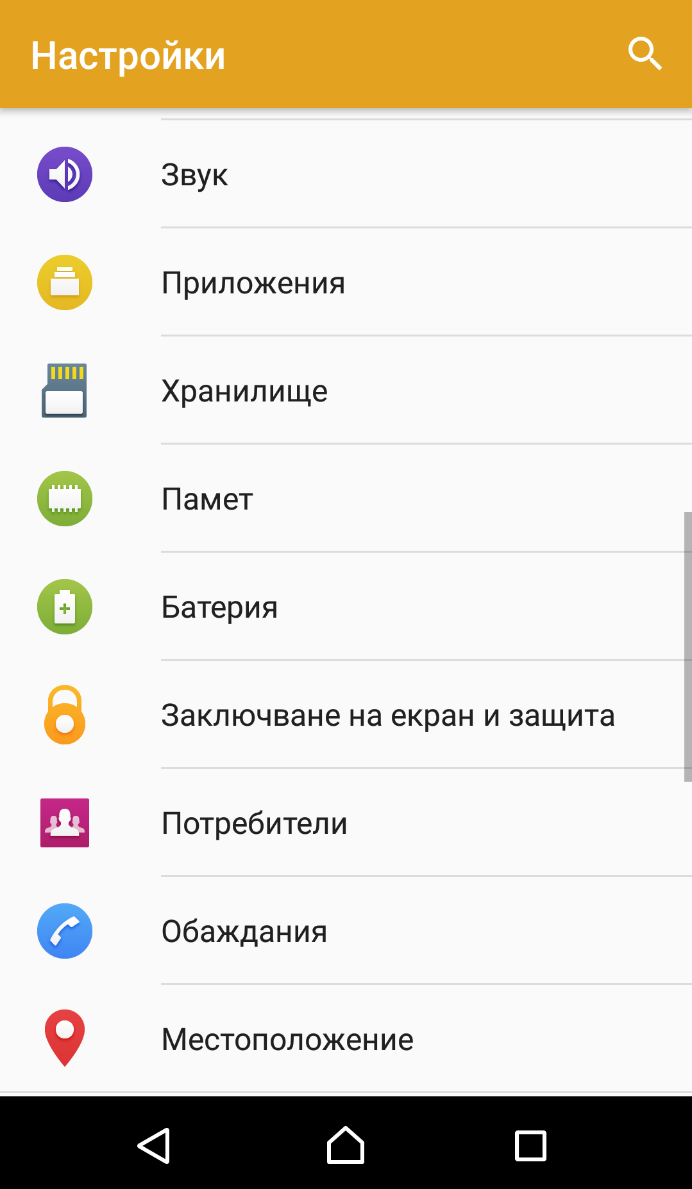
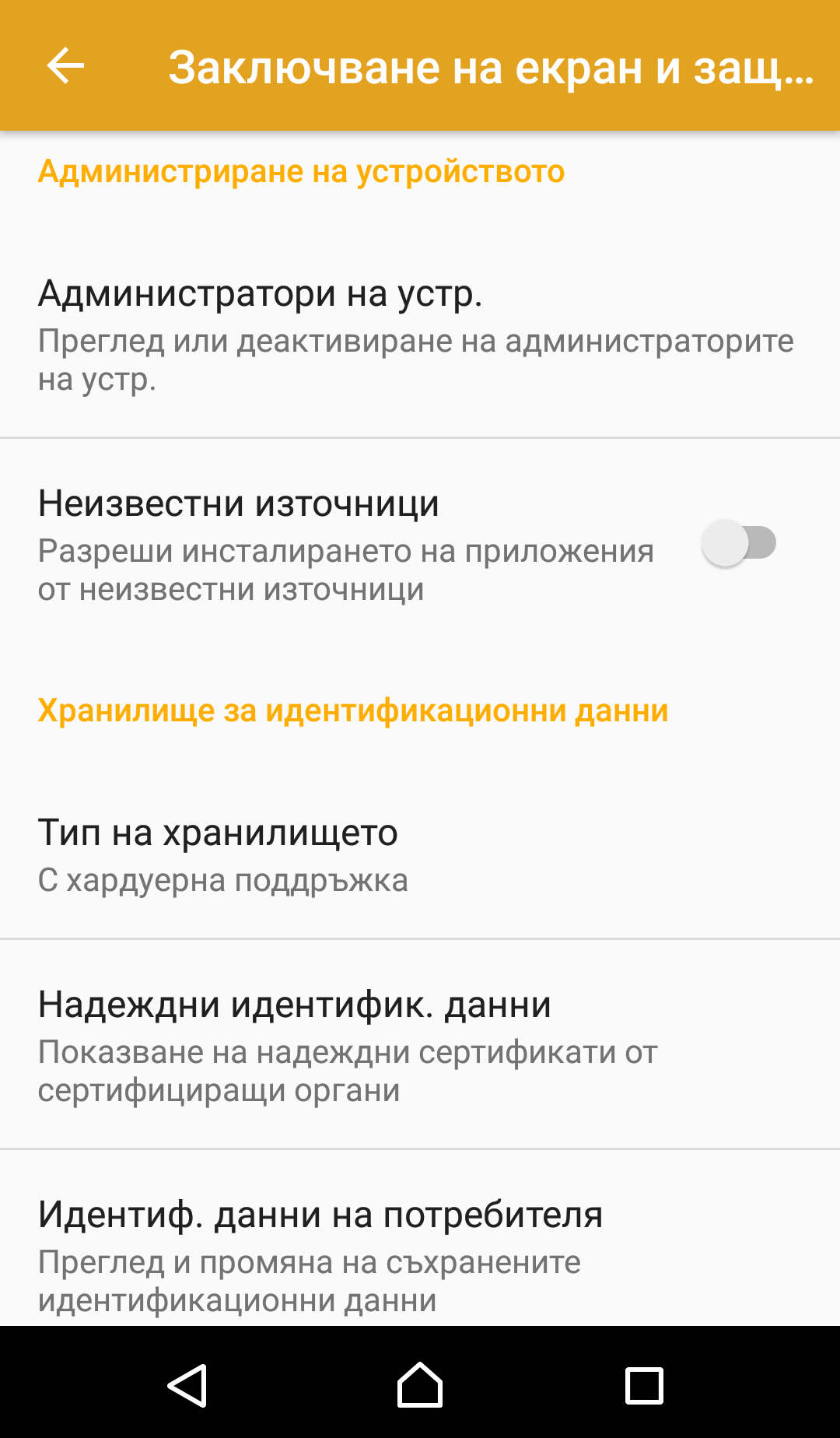
4.4 Изисквания към мобилната конфигурация

За да може един потребител да използва Android приложението, за управлението на колата, той трябва да разполага със смартфон, таблет и друг тип устройства, които работят с операционната система Android. Тъй като при писането на приложението е използван приложно-програмният интерфейс (на английски: API) 23, версията на операционната система трябва да е по-съвременна от 6.0.

4.5 Инструкции за инсталиране на Android приложението

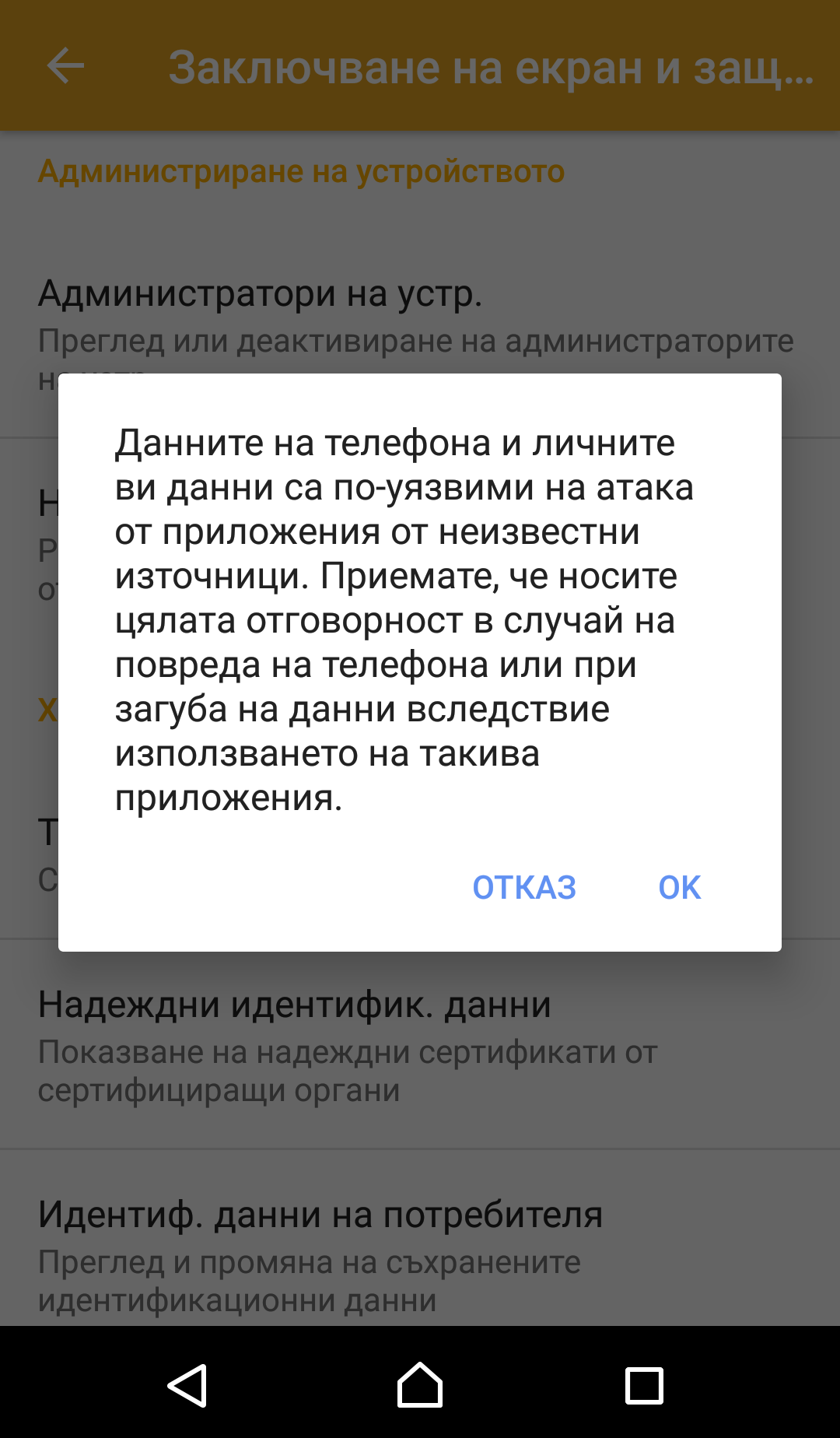
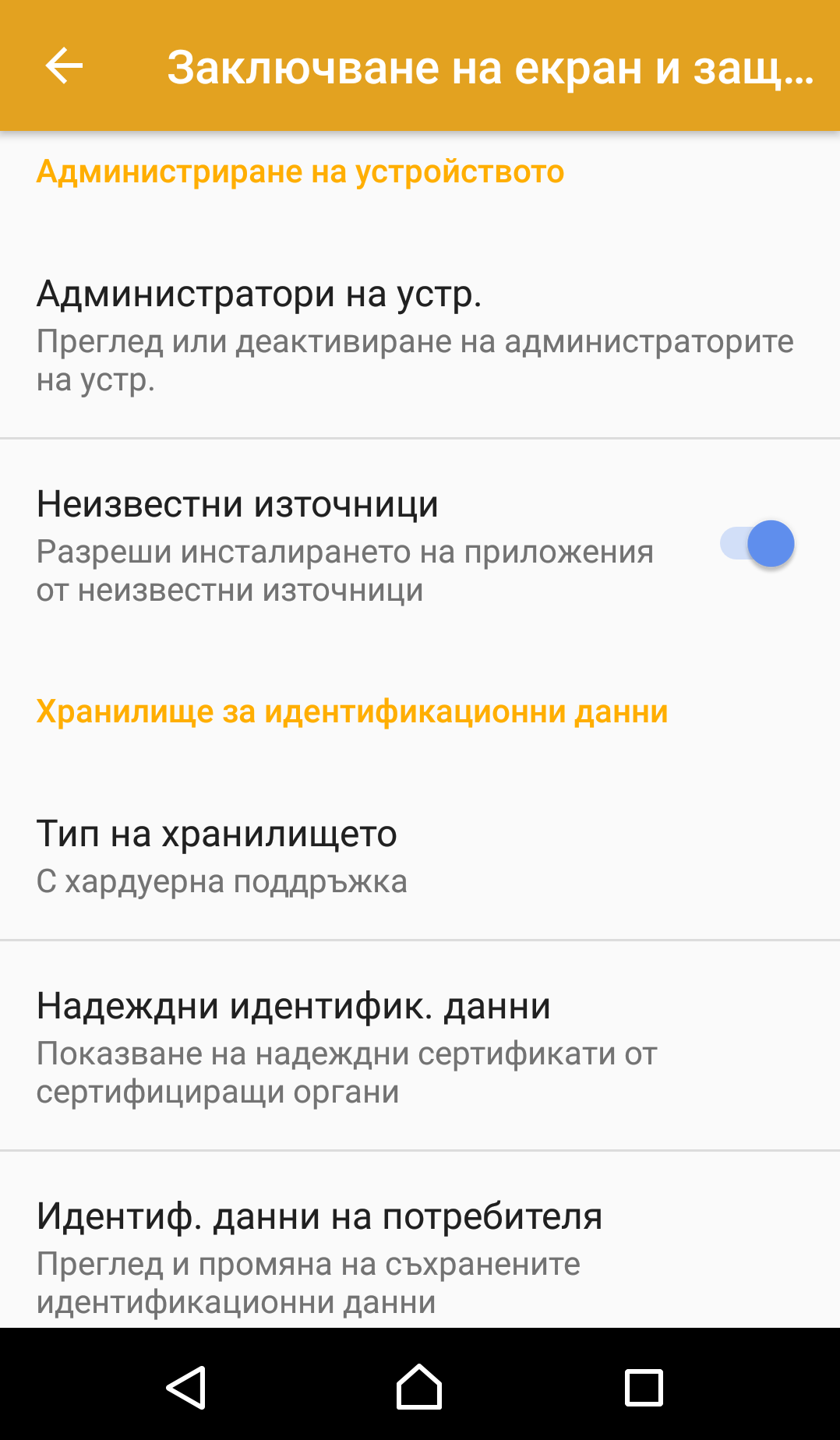
Android приложението трябва да бъде изтеглено. То ще е в стандартния за тези приложения - .apk файлов формат. Може да бъде изтеглено от Github хранилището на проекта -> <https://github.com/Jorkata12/RemoteControlCar>

**-** Тъй като приложението не е свалено от интернет магазина - Google Play Store, трябва специално да се позволи инсталирането на приложения от неизвестни източници. Това се прави от менюто Настройки >> Заключване на екран и защита >> Неизвестни източници.



***Фиг 4.4 и 4.5 Меню Настройки >> Заключване на екран и защита***

След натискане на бутона, отдясно на „Неизвестни източници“, се появява едно формално предупреждение, чийто текст е изписан на следващата фигура.



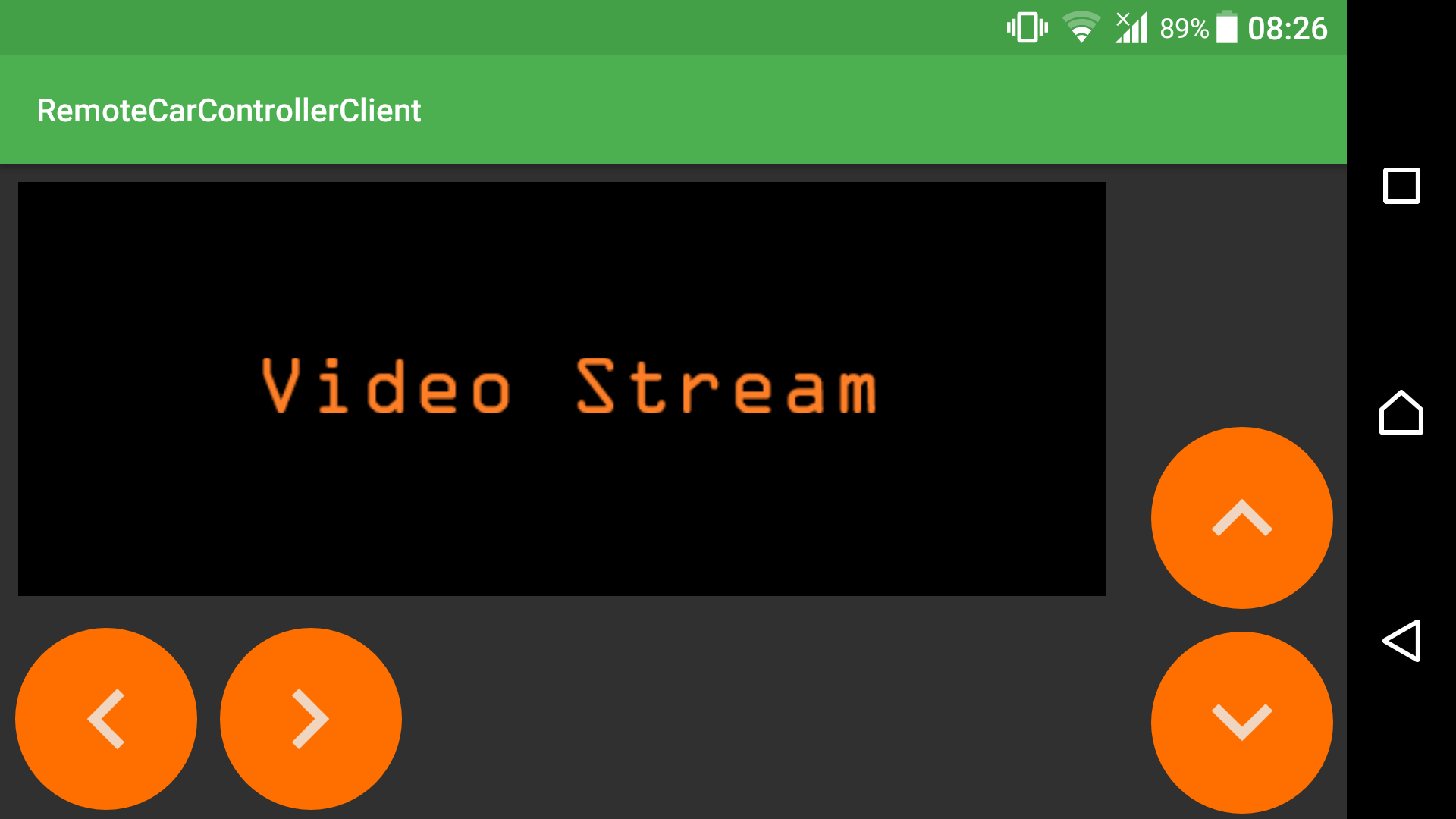
***Фиг 4.6 и 4.7 Предупреждение за инсталиране от неизвестни източници >> включен режим***

Инсталирането на приложението отнема малко време, но за да е възможно самото инсталиране е задължително устройството да разполага с файлов мениджър.

4.6 Използване на приложението

Android приложението изпълнява роля на дистанционно управление, с помощта на което се управлява колата. В следващите точки ще бъде описано как се използва приложението.

След като приложението бъде стартирано, се отваря екранът за управление на колата.



***Фиг 4.8 Екранът за управление на колата***

Както се вижда на фигурата, екранът има четири бутона, с които се управлява колата. Всеки бутон си има стрелка, указващата посоката на движение, като по този начин се улеснява потребителя.

Двата бутона, разположени вертикално, са за задвижване на колата напред/назад, а другите два хоризонтално разположени бутона, са да задвижване на колата наляво/надясно.

Освен бутоните има и екран, на фигурата наречен като Video Stream, на който може да види получения видео поток от сървъра. С това видео, потребителят може да вижда, както се случва пред количката докато е в движение.

ЗАКЛЮЧЕНИЕ

С настоящата дипломна работа беше разработен прототип на роботизирана кола, която може да бъде дистанционно управлявана от смартфон или всяко подобно устройство, използващо Android операционна система. В началото на документацията беше направен преглед на вече съществуващи роботизирани системи, популярни хардуерни платформи и среди за разработка на софтуер.

В проекта бяха реализирани следните функционалности:

- Управляващият хардуер на радио-управляема кола, беше подменен с едночиповата компютърна система – Raspberry Pi 3.

- Беше осъществена безжична WiFi връзка между Android устройството и колата.

- Комуникацията между двете устройства и управлението на колата са реализирани със стандартни HTTP заявки, изпращани от Android устройството към Java приложение, използващо Spring Boot и Apache Tomcat сървър.

- Свързване на камера към развойния хардуер, която да осигурява картина, показваща случващото се пред колата по време на движение.

Управляващото приложение предоставя един опростен потребителски интерфейс, като по този начин контролирането на колата се осъществява изключително лесно. Използването на популярни платформи, като Java предоставя една независимост откъм характеристиките на компютърната конфигурация, стига тя да има инсталиран Java Runtime Environment.

След всичко представено дотук, може да се отбележи, че проекта е един завършен прототип на роботизирана кола, който може да намери приложение, както в ежедневието на човека, така и в индустрията. Като бъдещи разработки могат да бъдат добавени различни сензори върху корпуса на колата, например – сензор за влажност, температура, разстояние.

ИЗПОЛЗВАНА ЛИТЕРАТУРА

1. *Robot*

[*http://www.newworldencyclopedia.org/entry/Robot*](http://www.newworldencyclopedia.org/entry/Robot)*, 15 Юли 2015, достъпена на 04 Февруари 2018*

1. *stackoverflow.com, потребител John E Harriss на дата May 28 '15 at 5:46, със следната връзка*

[*https://stackoverflow.com/questions/17308956/differences-between-hard-real-time-soft-real-time-and-firm-real-time*](https://stackoverflow.com/questions/17308956/differences-between-hard-real-time-soft-real-time-and-firm-real-time)*, последно достъпен на 04 Февруари 2018*

1. *Unmanned Systems*

[*https://www.qinetiq-na.com/products/unmanned-systems/*](https://www.qinetiq-na.com/products/unmanned-systems/)*, 2016, достъпена на 05 Февруари 2018*

1. *Dragon Runner*

[*https://www.qinetiq-na.com/products/unmanned-systems/dragon-runner/*](https://www.qinetiq-na.com/products/unmanned-systems/dragon-runner/)*, 2016, достъпена на 05 Февруари 2018*

1. *Dragon Runner™ 10 Micro Unmanned Ground Robot Datasheet*

[*https://www.qinetiq-na.com/wp-content/uploads/Datasheet\_DR10.pdf*](https://www.qinetiq-na.com/wp-content/uploads/Datasheet_DR10.pdf)*, 2016, достъпен на 05 Февруари 2018*

1. *Dragon Runner™ 20 Small Unmanned Ground Robot Datasheet*

[*https://www.qinetiq-na.com/wp-content/uploads/datasheet\_DR20\_lores.pdf*](https://www.qinetiq-na.com/wp-content/uploads/datasheet_DR20_lores.pdf)*, 2016, достъпен на 05 Февруари 2018*

1. *Tactical Robot Controller*

[*https://www.qinetiq-na.com/products/unmanned-systems/trc/*](https://www.qinetiq-na.com/products/unmanned-systems/trc/)*, 2016, достъпена на 05 Февруари 2018*

1. *Tactical Robot Controller Datasheet*

[*https://www.qinetiq-na.com/wp-content/uploads/data-sheet\_trc\_web1.pdf*](https://www.qinetiq-na.com/wp-content/uploads/data-sheet_trc_web1.pdf)*, 2016, достъпен на 05 Февруари 2018*

1. *Arduino*

[*https://en.wikipedia.org/wiki/Arduino*](https://en.wikipedia.org/wiki/Arduino)*, последно редактирана на 08 Февруари 2018, в 04:10, достъпен на 08 Февруари 2018*

1. *Raspberry Pi*

[*https://en.wikipedia.org/wiki/Raspberry\_Pi*](https://en.wikipedia.org/wiki/Raspberry_Pi)*, последно редактирана на 07 Февруари 2018, в 21:27, достъпен на 08 Февруари 2018*

1. *What is Arduino?*

[*https://www.arduino.cc/en/guide/introduction*](https://www.arduino.cc/en/guide/introduction)*, 2018, достъпена на 08 Февруари 2018*

1. *Raspberry Pi Specifications*

[*https://www.raspberrypi.org/magpi/raspberry-pi-3-specs-benchmarks/*](https://www.raspberrypi.org/magpi/raspberry-pi-3-specs-benchmarks/)*, 2017, достъпена на 09 Февруари 2018*

1. *Raspbian*

[*https://www.raspberrypi.org/documentation/raspbian/*](https://www.raspberrypi.org/documentation/raspbian/)*, 2018, достъпена на 09 Февруари 2018*

1. *Arduino Build Process*

[*https://github.com/arduino/Arduino/wiki/Build-Process*](https://github.com/arduino/Arduino/wiki/Build-Process)*, 09 Юли 2017, достъпена на 10 Февруари 2018*

1. *IntelliJ IDEA Features*

[*https://www.jetbrains.com/idea/features/*](https://www.jetbrains.com/idea/features/)*, 2018, достъпена на 10 Февруари 2018*

1. *JetBrains Plugins*

[*https://plugins.jetbrains.com/*](https://plugins.jetbrains.com/)*, 2018, достъпена на 10 Февруари 2018*

1. *IntelliJ Community Edition FAQ*

[*http://www.jetbrains.org/display/IJOS/FAQ*](http://www.jetbrains.org/display/IJOS/FAQ)*, 2018, достъпена на 10 Февруари 2018*

1. *Spring Framework*

[*https://projects.spring.io/spring-framework/*](https://projects.spring.io/spring-framework/)*, 2018, достъпена на 12 Февруари 2018*

1. *Dependency Injection*

[*https://angular.io/guide/dependency-injection*](https://angular.io/guide/dependency-injection)*, 2018, достъпена на 13 Февруари 2018*

1. *The IoC Container*

[*https://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/beans.html*](https://docs.spring.io/spring/docs/3.2.x/spring-framework-reference/html/beans.html)*, 2018, достъпена на 13 Февруари 2018*

1. *Spring Boot – Simplifying Spring for Everyone*

[*https://spring.io/blog/2013/08/06/spring-boot-simplifying-spring-for-everyone/*](https://spring.io/blog/2013/08/06/spring-boot-simplifying-spring-for-everyone/)*, 06 Август 2013, достъпена на 13 Февруари 2018*

1. *Microbot Motor Driver*

[*https://www.microbot.it/en/product/73/DC-Dual-Motor-Driver-30V-4A-V2.html*](https://www.microbot.it/en/product/73/DC-Dual-Motor-Driver-30V-4A-V2.html)*, достъпена на 13 Февруари 2018*

1. *Camera Module v2*

[*https://www.raspberrypi.org/products/camera-module-v2/*](https://www.raspberrypi.org/products/camera-module-v2/)*, 2018, достъпена на 13 Февруари 2018*

1. *Google Volley*

[*https://developer.android.com/training/volley/index.html*](https://developer.android.com/training/volley/index.html)*, 2018, достъпена на 04 Март 2018*

1. *Setting up a RequestQueue*

[*https://developer.android.com/training/volley/requestqueue.html*](https://developer.android.com/training/volley/requestqueue.html)*, 2018, достъпена на 04 Март 2018*

**СЪДЪРЖАНИЕ**

[УВОД 4](#_Toc508230231)

[ПЪРВА ГЛАВА 6](#_Toc508230232)

[ОБЗОР НА СЪЩЕСТВУВАЩИТЕ ПОДОБНИ СИСТЕМИ, КАКТО И МЕТОДИТЕ, ТЕХНОЛОГИИТЕ И РАЗВОЙНИТЕ СРЕДИ ЗА ТЯХНАТА РАЗРАБОТКА. ПРОУЧВАНЕ НА ЕЛЕМЕНТНА БАЗА 6](#_Toc508230233)

[1.1 Обзор на съществуващи подобни системи 6](#_Toc508230234)

[1.1.1 Робот QinetiQ North America Dragon Runner 10 6](#_Toc508230235)

[1.1.2 Робот QinetiQ North America Dragon Runner 20 8](#_Toc508230236)

[1.1.3 Tactical Robot Controller 10](#_Toc508230237)

[1.2 Преглед на хардуерни платформи 11](#_Toc508230238)

[1.2.1 Arduino 12](#_Toc508230239)

[1.2.2 Raspberry Pi 13](#_Toc508230240)

[1.3 Преглед на известни развойни среди при разработката на софтуер за вградени системи 14](#_Toc508230241)

[1.3.1 Arduino IDE 14](#_Toc508230242)

[1.3.2 IntelliJ IDEA 16](#_Toc508230243)

[1.3.3 Eclipse 17](#_Toc508230244)

[1.4 Преглед на известните операционни системи за мобилни устройства 18](#_Toc508230245)

[1.4.1 iOS 18](#_Toc508230246)

[1.4.2 Android 19](#_Toc508230247)

[ВТОРА ГЛАВА 20](#_Toc508230248)

[ОБЩИ ИЗИСКВАНИЯ И ПРОЕКТИРАНЕ НА ПРОГРАМНИЯ ПРОДУКТ. ИЗБОР НА ПРОГРАМНИ ЕЗИЦИ И СРЕДИ ЗА РАЗРАБОТКА 20](#_Toc508230249)

[2.1 Общи изисквания към програмния продукт 20](#_Toc508230250)

[2.2 Избор на език за програмиране на управляващия софтуер и мобилното приложение 20](#_Toc508230251)

[2.3 Избор на развойна среда при разработката на управляващия софтуер 22](#_Toc508230252)

[2.4 Избор на развойна среда при разработката на мобилното приложение 22](#_Toc508230253)

[2.5 Избор на основен развоен хардуер 23](#_Toc508230254)

[2.6 Spring Framework и проектът Spring Boot 24](#_Toc508230255)

[2.8 Използвани електронни елементи 26](#_Toc508230256)

[2.8.1 Microbot DC Dual H-bridge Motor Driver 27](#_Toc508230257)

[2.8.2 Raspberry Pi Camera Module 28](#_Toc508230258)

[ТРЕТА ГЛАВА 30](#_Toc508230259)

[ПРОЕКТИРАНЕ И РЕАЛИЗАЦИЯ НА РОБОТИЗИРАНАТА КОЛА И ПРИЛОЖЕНИЕТО ЗА УПРАВЛЕНИЕ 30](#_Toc508230260)

[3.1 Блокова схема на свързване на хардуерните елементи 30](#_Toc508230261)

[3.2 Електрическа схема 32](#_Toc508230262)

[3.3 Широчинно-импулсна модулация (ШИМ) 34](#_Toc508230263)

[3.4 Използвани анотации на Spring в проекта 35](#_Toc508230264)

[3.5 Сървърна част на проекта 36](#_Toc508230265)

[3.5.1 IP конфигурация на Raspberry Pi 36](#_Toc508230266)

[3.5.2 Библиотека за управление на входно-изходни операции върху Pi 37](#_Toc508230267)

[3.5.3 Управление на колата и крайни точки на приложението 37](#_Toc508230268)

[3.4.5 Инициализация на обектите от тип Motor 40](#_Toc508230269)

[3.4.6 Структура на класовете в приложението 43](#_Toc508230270)

[3.4.6.1 Класът MotorRepository 43](#_Toc508230271)

[3.4.6.2 Класът CarService 44](#_Toc508230272)

[3.4.6.3 Класът CarController 45](#_Toc508230273)

[3.6 Клиентска част на проекта 46](#_Toc508230274)

[3.6.1 Реализиране на шаблони за дизайн 46](#_Toc508230275)

[3.6.2 Библиотека за изпращане на HTTP заявки 47](#_Toc508230276)

[3.6.3 Класът RequestQueueSingleton 47](#_Toc508230277)

[3.6.4 Класът org.elsys.remote\_control\_car.request.Request 49](#_Toc508230278)

[3.6.5 Класът RequestFactory 50](#_Toc508230279)

[3.6.6 Класът OnTouchListenerImpl 52](#_Toc508230280)

[3.7 Получаване на видео поток от камерата към Android приложението 53](#_Toc508230281)

[ЧЕТВЪРТА ГЛАВА 55](#_Toc508230282)

[РЪКОВОДСТВО НА ПОТРЕБИТЕЛЯ 55](#_Toc508230283)

[4.1 Изисквания към компютърната конфигурация 55](#_Toc508230284)

[4.2 Необходими хардуерни елементи (Bill Of Materials) 55](#_Toc508230285)

[4.3 Инструкции за качване на уеб приложението на развойния хардуер 56](#_Toc508230286)

[4.4 Изисквания към мобилната конфигурация 58](#_Toc508230287)

[4.5 Инструкции за инсталиране на Android приложението 59](#_Toc508230288)

[4.6 Използване на приложението 61](#_Toc508230289)

[След като приложението бъде стартирано, се отваря екранът за управление на колата. 61](#_Toc508230290)

[ЗАКЛЮЧЕНИЕ 63](#_Toc508230291)

[ИЗПОЛЗВАНА ЛИТЕРАТУРА 64](#_Toc508230292)