УНИВЕРСИТЕТ ПО БИБЛИОТЕКОЗНАНИЕ И ИНФОРМАЦИОННИ ТЕХНОЛОГИИ

КАТЕДРА ”ИНФОРМАЦИОННИ СИСТЕМИ И ТЕХНОЛОГИИ”

МАГИСТЪРСКА ПРОГРАМА

”СОФТУЕРНО ИНЖЕНЕРСТВО”

МАГИСТЪРСКА ТЕЗА

на тема:

Анализ и изследване на системи за автономно управление на автомобили и дефиниране на минимален набор от модули за функционираща автономна автомобилна система

Дипломант:

Александър Василев

задочно обучение

Ф.№ 016-сиз

Научен ръководител:.....

(проф. И. Гарванов)

София

2016

РЕЗЮМЕ

Василев, А. Анализ и изследване на системи за автономно управление на автомобили и дефиниране на минимален набор от модули функциониращa автономна система. Научен ръководител проф. И. Гарванов. София 2016. Катедра «Информационни системи и технологии». Магистърска програма

”Software Engineering”. УНИБИТ. 85 с. Брой източници – 38, приложения – 0.

Целта на дипломната работа е да разгледа вече налични системи за автономно управление на наземни превозни средства и да дефинира минимална имплементация на такава система, като за целта ползва технологии като Bluetooth LE и RS232 за комуникация, SD card за четене и записване на данни, както и GPS и електронен компас за навигация. В магистърската теза се разглеждат налични имплементации, съдържа се описание на ползваните технологии в една примерна система, съдържаща минималните модули нужни за изпълняване на целта ѝ; разписани са примерни софтуерни модули и са направени някои тестове с прототипирания хардуер и софтуер.

Като резултати е проектирана минимална система за автономен автомобил, като са създадени отделни прототипи на нужните модули, с цел получаване на измервания.

Ключови думи: autonomous driving, smart car.

СЪДЪРЖАНИЕ

УВОД

Автономното управление на автомобили и друг вид техника става все по-популярна тема. Водещи автомобилни компании инвестират огромни средства за създаване на самоуправляващи се автомобили, които могат безопасно да се движат по улиците, без нуждата за някаква допълнителна пътна инфраструктура. Вече разработени, автоматично управляеми системи има и в сферата на агро-бизнеса (автономни машини - трактори за разкопаване, наторяване и полагане на химикали, както и придружаващи ги камиони, комбайни и др.), самоуправляващи се дронове, малки безпилотни самолети и др. Всички тези системи трябва да имат имплементирани, както системи за изчисление и навигация по маршрут, така и fail-safe системи, които да им позволяват да реагират адекватно при неочаквани ситуации или повреди в оборудването им (безопасно изключване, приземяване, спиране и т.н.). Развиването на тези автономни устройства позволява до голяма степен задачите, изпълнявани от тях да бъдат извършени с голяма прецизност и да бъдат направени по най-ефективния начин.

Автономните автомобили също така, биха подпомогнали за намаляването на инцидентите по пътищата. Някои автомобилни компании дори смятат, че интелигентните системи в автомобилите имат потенциала напълно да предотвратяват всички фатални пътни инциденти, поради бързата изчислителна мощ на новите интегрирани компютърни системи [1]. Това става благодарение на множество сензори, поместени в превозните средства, които позволяват на вътрешните компютри да анализират състоянието и обстановката на пътя в реално време във всички посоки около превозното средство. По този начин обстановката може да бъде анализирана многократно по-добре, от колкото би могъл да я възприеме човек и това би позволило превозното средство да взима по-адекватни мерки, осигуряващи безопасността на участниците в движението.

Актуалността на темата е подчертана от изброените по-горе причини, като пазарът за такава техника се очаква да расте. Прогнозите са такива, че до 2020 година би

трябвало да има около 10 милиона автомобила с функция за автоматично управление [2]. Голямата актуалност на темата, както и факта, че се изискват познания по софтуерно инженерство са мотивите за избора й.

Предмета на магистърската теза е подбирането на хардуер и софтуер, който да е минимума за една проста автономна система за управление на електрическа кола по предварително зададени координати. Това ще бъде постигнато чрез използване на различни технологии за ориентиране и комуникация с устройства, позволяващи управлението на електрическата кола. Също така биват разгледани някои тенденции и съществуващи имплементации на подобни системи.

I. ГЛАВА ПЪРВА: СЪЩЕСТВУВАЩИ АВТОНОМНИ СИСТЕМИ, ИЗПОЛЗВАНИ ТЕХНОЛОГИИ И ПРАКТИКИ

1 Какво представляват автономните системи за управление на машини

1.1 Дефиниции

1.1.1 Автономен автомобил – това е автомобил, който има възможността да приема информация за околната среда чрез сензори и да се навигира през нея без човешка намеса за командите на движение. Най-често автономните автомобили засичат околната среда и препятствията чрез технологии като радар, Lidar, GPS, компютърна визия и др. След това сложни системи за управление интерпретират събраната информация от сензорите и изчисляват подходящи пътища, както и възможно препятствия по тях. [3]

1.1.2 Микроконтролер – Това е малък компютър (SoC или System on a Chip), побиращ се на интегрална схема, който съдържа процесорно ядро, памет, програрируеми входо-изходни периферии (може да са просто input-output пинове, аналогово-дигитални конвертори, хардуер за серийна комуникация и др.). [4]

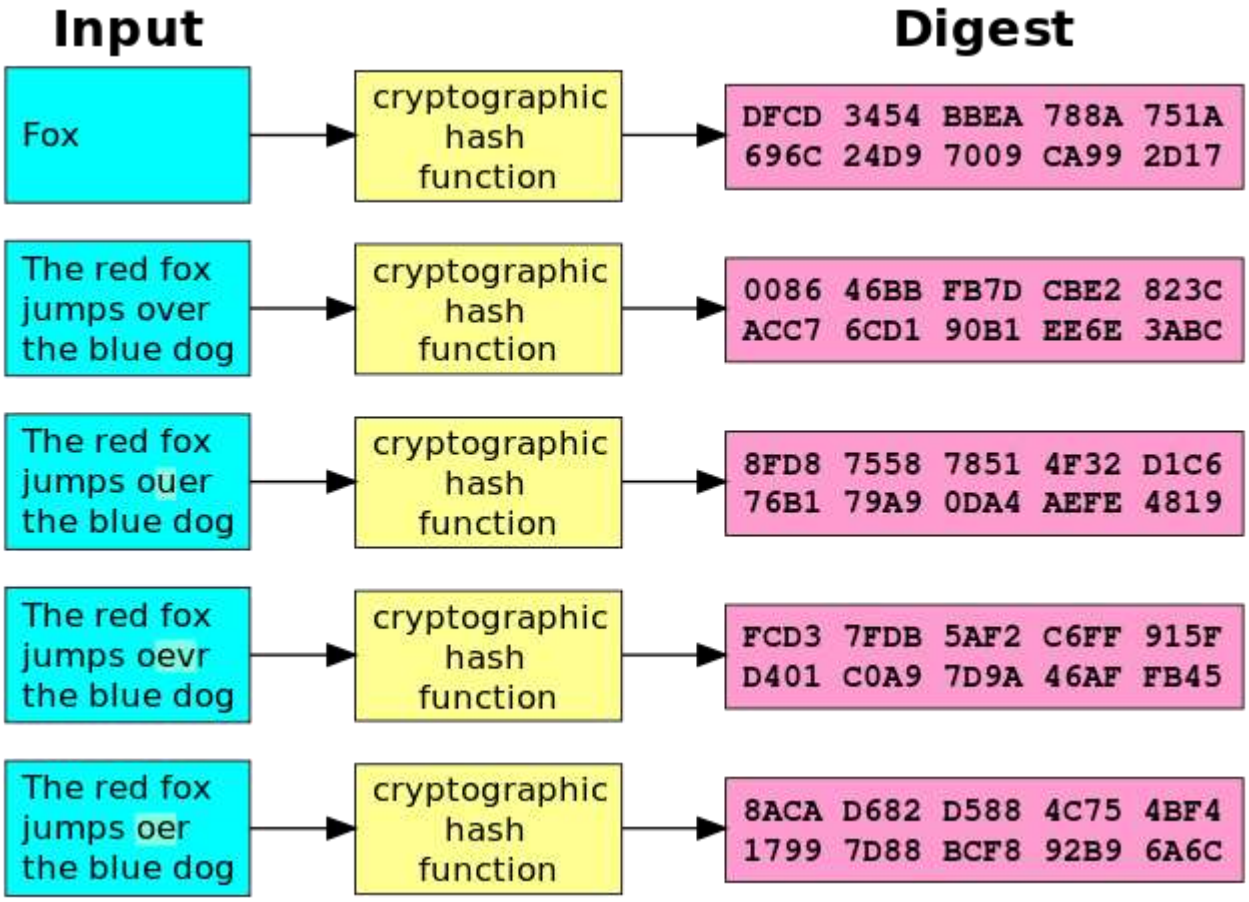
1.1.3 Флашване – Флашването представлява записване на програмен код или данни, които трябва да бъдат запазени в модули на EEPROM или флаш памет. Това се прави цел ъпдейтване на програмния код или презаписване на нови конфигурационни стойности. [5]

1.1.4 Hash Функция – Hashфункция в криптографията представлява функция, която практически е невъзможна да бъде обърната – тоест да се пресъздадат входните данни, имайки налични единствено изходните данни. Идеалната криптографическа хеш функция има следните четири главни свойства [6]:

- Хешът на всяко входно съобщение може да бъде изчислен бързо.
- Неосъществимо е да бъде пресъздадено входното съобщение от резултат на хеш функция
- Неосъществимо е да се промени входното съобщение без да се променят изходните данни от хеш функцията

- Неосъществимо е да се намерят две различни входни съобщения с една и съща хеш стойност. [6]

На следващата фигура е илюстрирано действието на хеш функция:



Фигура 1 [<wikicryptohashfunction>]

Както се вижда на Фигура 1 [<wikicryptohashfunction>] резултата от хеш функциите винаги е с еднага големина, независимо от входното съобщение. Също така дори при малки разлики във входните съобщения в резултата се наблюдават драстични разлики.

1.2 Примери

1.2.1 Tesla

Електрическите автомобили на Tesla са едни от първите, които могат да се управляват сами, но технологията е полу-автономна. Нужно е водача да въведе крайна точка и да предаде управлението на колата. Интересното при системата на Tesla е, че статистически данни за всяко пътуване се изпращат до централизирана

сървърна мрежа, която анализира всички резултати и специална невронна мрежа ги обработва. По този начин управлението на автомобила има възможността да се обучава само, като софтуера на колата се обновява автоматично по безжичен път (чрез мобилна мрежа или WiFi). [7]

Докато се управлява сама обаче, на шофьорската седалка задължително трябва да има водач, който да се намеси при екстремни ситуации при които автомобила не може да реагира адекватно. Също така системата може да бъде задействана само по време на пътуване, когато автомобила е засякъл, че ситуацията е безопасна за преминаване към автономно управление. [7]

Както всяка система обаче и полу-автономната система за управление на автомобил не е съвършена. В някои случаи собствениците се оплакват, че колите предприемат неадекватни маневри – навлизане в завои с прекалено висока скорост, грешна линия при навлизане на завои, дори и опасни маневри, които ако не са били предотвратени са щели да доведат до челен сблъсък с коли в насрещното движение. [8]

Благодарение на безжичните ъпдейти на софтуера и бързото самообучение на алгоритъма за управление, Tesla се надява да направи своя автопилот много по-безопасен в близкото бъдеще и да не позволява на шофьорите да прилагат опасни настройки по време на пътуване. Също така се работи и по функции, които ще позволят на автомобилите да се паркират сами в гаражите на собствениците си, да намират станции за бързо зареждане, до които да се придвижват сами, да взимат собствениците си от дадено местоположение и дори да се паркират на предварително определено местоположение в зададен час (синхронизация с календар). [9]

1.2.2 Google

В края на 2009 – началото на 2010 година Google стартира своя проект за автономен автомобил. От тогава до сега автономните автомобили на Google включват модифицирани Toyota Prius, Lexus RX и през 2014 напълно нова кола, разработена от Google да бъде автономно управлявана. Това е и основната разлика от

автомобилите на Tesla – в Tesla автопилотът е разработен в последствие и има за цел да подпомага водача с управлението на колата, докато автономния автомобил на Google е разработен от самото начало да се управлява изцяло сам. За целта са премахнати дори и уредите за управление на конвенционален автомобил от самия интериор – липсват волан и педали.

1.2.2.1 Статистики

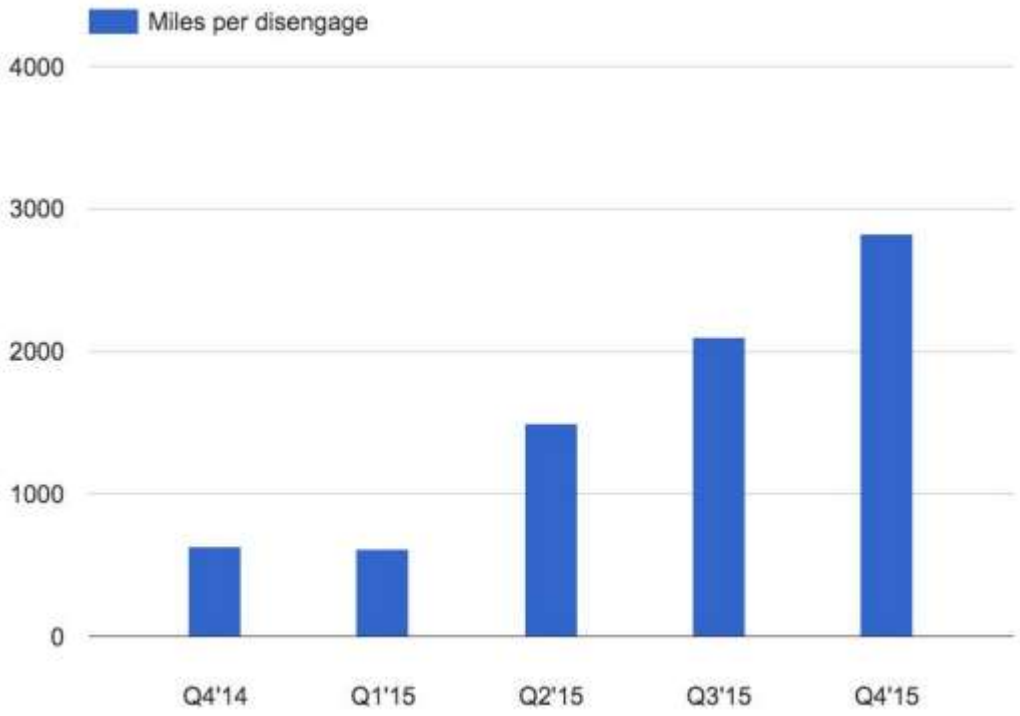
Според докладите на Google, към декември 2015 година техните самоуправляващи се автомобили са изминали над 1 372 111 мили от началото на проекта през края на 2009 година. Също така към момента правят по от 10 000 до 15 000 тестови мили всяка седмица, като целта е да се натрупа голям опит и да се отстранят възможно най-много проблеми в най-скоро време. [10]

От Google са публикували и статистически доклад относно броя на случаите, в които хората вътре в тестовите автомобили е трябвало да реагират поради повреди в хардуера, грешки в софтуера, застрашителни маневри от самите автономни автомобили или други участници в движението, като средното време за реакция на тестовите шофьори било 0,7 секунди. [11] Статистиката за броя аварийни намеси може да бъде видяна на Таблица 1, докато на Фигура 2 може да бъде видяно как нараства изминалото разстояние на което се налага аварийна намеса.

Месец	Аварийни намеси	Мили автономно шофиране
2014/09	2	4207.2
2014/10	19	23871.1
2014/11	21	15836.6
2014/12	43	9413.1
2015/01	53	18192.1
2015/02	14	18754.1
2015/03	30	22204.2
2015/04	51	31927.3
2015/05	13	38016.3
2015/06	11	42046.6
2015/07	29	34805.1
2015/08	7	38219.8
2015/09	16	36326.6

2015/10	16	47143.5
2015/11	16	43275.9
Общо:	341	424331

Таблица 1 [11]



Фигура 2 [11]

Според статистическите данни все по-рядко се налага на шофьорите да се намесват в работата на автономните автомобили. Също така и факта, че всеки месец биват изминавани все повече мили от автономните автомобили означава, че се трупа все по-голям опит и все повече проблеми биват поправяни, което ще доведе до запазване на тенденцията от Фигура 2. [<alphabetautonyeardis>]

1.2.2.2 Ползвани технологии

Автомобилите на Google обработват картографически данни и данни от сензори за да се навигират по пътищата. Технологията позволява на автомобила да знае къде точно се намира на картата и в коя лента на движение по пътя.

Софтуера класифицира всички засечени обекти на пътя – хора, колоездачи, други автомобили и препятствия, базирано на тяхната форма, големина и модели на движение. След това софтуера предвижда какво може да направят засечените

обекти – например ако човек се готви да пресече платното за движение, като реагира адекватно – намалява скоростта, предприема избягваща маневра и т.н. [12]

Автомобилите на Google техника на борда, кото струва около 150 000$, като около 70 000$ от тях струва и най-важния модул – LIDAR. Така с помощта на лазерно сканиране автомобила може да създаде виртуална 3D картина на заобикалящата го среда, която може да стигне чак до две футболни игрища напред. По този начин и с помощта на камерите могат да бъдат различавани различните обекти и тяхната посока и движение. Интересното в системата е, че поради огромния поток данни, някои от тях се изпращат дистанционно до сървърни ферми за да бъдат обработени. [12]

Недостатъка на тази система е, че не работи много добре в лошо време. Намалената видимост и допълнителните отражения от мократа повърхност не влияят добре на работата на сензорите, като за купола в който се намират е трябвало дори да бъдат направени специални чистачки, които да го чистят от замърсяване и капки дъжд. [13]

1.2.3 Автономни Трактори

Автономните машини са навлезли и на пазара за земеделска техника. В днешно време има трактори, които са полу-автономни или напълно автономни и могат сами да извършват земеделски задачи.

1.2.3.1 История

Идеята за автономен трактор се ражда още през ранната 1940 година, когато Frank W. Andrew измисля свой собствен. За да насочва своя автономен трактор бил използван варел в средата на полето, около който се навива кабел, закачен на самия трактор – по този начин трактора се движел в кръгове, които ставали все по-малки. През 1950 година компанията Ford разработила трактор без шофьор, наричан “The Sniffer”, но той никога не влезнал в производство. Причината за това е, че за да работи под самото поле трябвало да има прекарани специални жици.[14]

През следващите години нямало големи открития в сферата на автономните трактори чак до 1994 година, когато инженери от Silsoh Research Institute разработили система за анализ на изображения. Тя позволила управлението на малки автономни трактори. Този нов трактор можел дори да обръща като достигне края на полето, за да започне работа по нова линия. [14]

1.2.3.2 Съвременни разработки

a) През 2004 година учени от The Royal Veterinary and Agricultural University (KVL), отделение за Агрикултурни Науки в Дания разработват система за автономно управление на трактор, като отделят особено голямо внимание на сигурността.

   За целта те взимат средно голям трактор от Hako-Werke GmbH и го модифицират за автономна работа. Към него добавят готови продукти като:

   RTK GPS система за определяне на координатите. [15]

   Система за автоматично изчисление на маршрута на агрикултурни машини (предимно трактори). Системата е модел AgroNav GT2000 на компанията GEOTEC (фалирала). Тази система представлява вграден компютър с плосък екран с клавиши около него. Върви под операционната система Microsoft Windows 2000 и по подразбиране изпълнява навигационната програма. Системата също така може да ползва USB клавиатура, като планираните навигационни пътища се записват на usb флаш памет. GPS приемника изпраща данните към GT2000 системата посредством стандартен RS232 интерфейс. Направени са и промени по завиването и управлението на CVT трансмисията, за да могат да бъдат контролирани от компютър чрез пневматика. Към цялостната система е закачен и допълнителен компютър, който да управлява системите за сигурност на автономния трактор, чрез които той може да спре от пълна скорост до пълен покой в рамките на 2 метра. [15]

   Системите за безопасност са нужни, понеже въпреки, че машината не се движи в силно населени места все пак може да попадне на препядствия, хора

или животни по планирания маршрут. Също така системата разчита да приема постоянни сигнали от човек, който натиска така наречения Dead Man’s Switch – бутон, който ако бъде отпуснат машината спира работа. [15]

Системата обаче се оказала непрактична за имплементация, понеже за един трактор са били нужни двама души, които да го обслужват – един оператор и един техник по сигурността. [15]

Като резултати от изпитанията обаче, системата показала изключителна точност при управление. Компютърът успял да управлява трактора по полетата с точност около 3см от планирания маршрут. [15]

b) John Deere

John Deere е една от компаниите с най-голямо въздействие в агрикултурната индустрия, поради лидерската си позиция. Към момента имат прототип и разработват активно автономни трактори. Техните машини използват куполни антени за да приемат GPS координати и радио сигнали от оператори, когато комуникацията с GPS сателитите е възпрепятствана. [14]

c) Autonomous Tractor Corporation

ATC предлагат автономен трактор наречен SPIRIT. От начало компанията произвеждала полу-автономни трактори (такива, които се движат зад друг трактор управляван от шофьор, следвайки точно неговите движения), като тенденцията е да предлага изцяло автономни решения. Те представили своите изцяло автономни трактори още през 2012 година. [16]

Тракторите са задвижвани от 2 двигателя, които служат за генератори, които от своя страна задвижват 4 електрически мотора – по един на всяко от колелата на трактора. [16]

За навигация по полетата тракторите не използват GPS система, а виртуални карти на терена с цел по-точното позициониране и избягване на препятствия. Виртуалните карти се изграждат посредством лазери и радио вълни. В трактора има общо 10 компютъра – 2 за навигация, 2 за контролиране на

компонентите на трактора, 4 за управление на електрическите мотори и 2 за хидравличната система. Навигационните компютри и тези за управлението непрекъснато обменят информация, като ако не са изчислили една и съща позиция, на която мислят, че се намира трактора, то машината спира автоматично. Ако се наложи автоматично спиране, машината ще попита човек-оператор какво да прави след това. Системите, които са инсталирани на трактора са готови, доказани системи, които се ползват повече от 20 години в северния Атлантически океан за управление на кораби, обслужващи нефтени платформи. [16]

Системата също така позволява и полу-автономен режим на работа, където трактора-робот следва движещ се пред него трактор, управляван от човек. [16]

Като заключение може да бъде обобщено, че повечето производители на трактори първо смятат да преминат през полу-автономни машини, а не директно към напълно автономни. Това се отнася и за големи компании като CASE International Harvester, John Deere, Fendt и др. Чрез полу-автономните им технологии те целят да съберат достатъчно данни за направата на подробни дигитални модели и изграждане на коректно поведение на автономните машини, което може да покрие повечето ситуации, които биха възникнали в реални условия на експлоатация. Тенденцията е да се обединят системите за прецизна агрикултура (Precision agriculture) и автономно управление – прецизната агрикултура ще изчислява оптималните пътища на машините и количеството материал, което трябва да използват за обработка на земята, докато машините се управляват сами и изпълняват предварително изчислените планове. Това би довело до увеличение на продукцията, тъй като в момента има недостиг на квалифициран персонал, намаляване на разходите (особено, ако бъдат ползвани трактори с електро мотори за задвижване на колелата) и увеличение на печалбата (по оптимално изразходвани ресурси и намаляване на работническата ръка, поради автоматизирането на процесите).

1.2.4 Военни разработки за наземни машини.

Военното министерство на САЩ има активен интерес в разработката на наземни автономни машини още от 2004 година, когато DARPA организира DARPA Grand Challenge. Целта на събитието е частни организации да разработят наземни автономни машини, които трябва да стигнат до определена цел през труднодостъпен терен, като за първите финиширали има парична награда. В първата година на състезанието нито една разработка не успява да стигне до финалната линия, но следващите години са по-успешни. [17]

Това мероприятие дава основите на съвременните разработки и проекти на DARPA. Още през 2012 година Oshkosh Corporation има работещ прототип на автономен 15 тонов военен камион, който може сам да се справя с труднодостъпен терен. [18]

През 2014 година Lokheed Marting успешно провежда изпитания на тяхна автономна система за тежки наземни машини. Автономно управляваните камиони успешно изпълняват зададените мисии, като между тях се движат и автомобили, управлявани от хора. Това изпитание е имало за цел сформирането на конвой между няколко военни машини. Камионите също така успешно се навигират и сред жилищни сгради. [19]

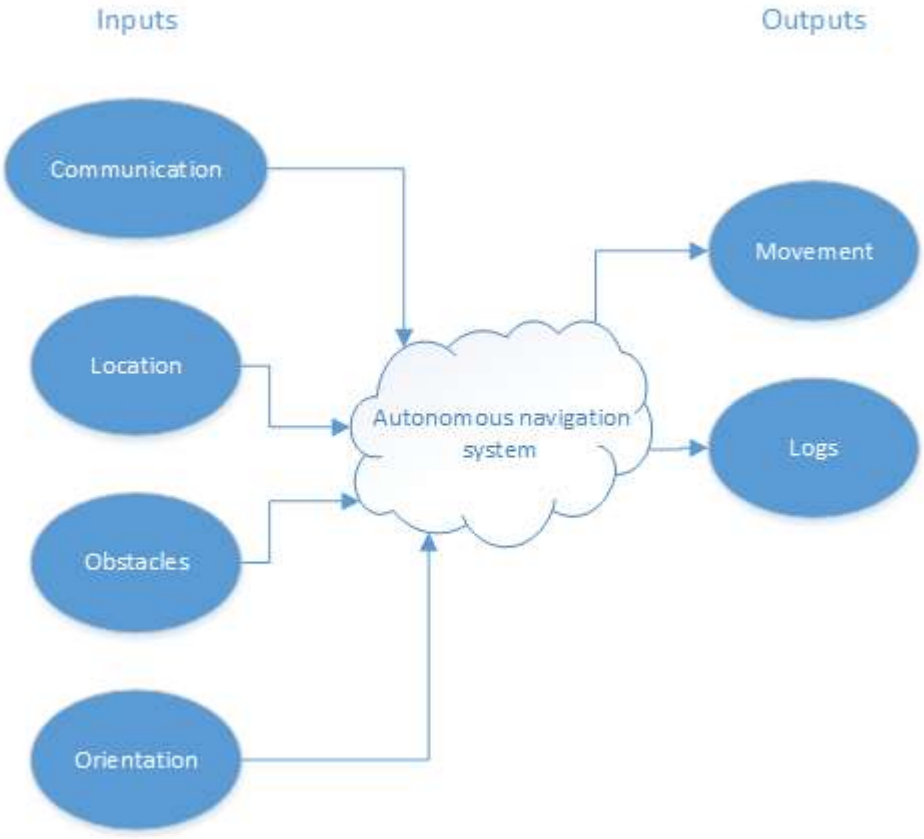
Като допълнение армията на САЩ е планирала тестове, които ще бъдат проведени през лятото на 2016 година. Те ще целят преминаването на конвой по магистрала, част от публичните пътища и ще ползват комуникация тип превозно средство – към – инфраструктура и превозно средство – към – превозно средство. Тази комуникация ще има за цел предаването и споделянето на данни от инфраструктурата към движещите се машини, която ще представлява състояние на настилката, ограничения на скоростта и др. За да се постигне това на самата автомагистрала ще бъдат инсталирани специални модули с транспондери, които имат комуникационен обхват от 300 метра, като всеки един струва около 5000 щадски долара. Като допълнение всяко превозно средство от конвоя има

допълнително оборудване на стойност около 179 000 щатски долара, което включва LIDAR сензори, радари, камери и др.[20]

Целта на американските военни е с въвеждането на автономните наземни машини до голяма степен да се намали човешката намеса при логистичните им мисии. По този начин ще се осигури по-голяма сигурност (липса на хора в превозни средства, които биха били цел на атаки), по-точно и ефективно изпълнение на мисиите и намаляване на разходите.

2 Избран минимален набор от технологии за автономна система

За да може да функционира една автономна система за навигация, тя има нужда от следните входове (изобразени от лявата страна във фигурата) и произвежда съответните изходи (изобразени от дясната страна във фигурата):



Фигура 3 (Общ поглед на системата)

2.1 Комуникационни

Комуникационните технологии в една автономна система са важни, понеже чрез тях се задават целта, записват се конфигурации и може да бъде следена системата в

действие и при нужда изключвана. Избрани са безжични и кабелни технологии за комуникация.

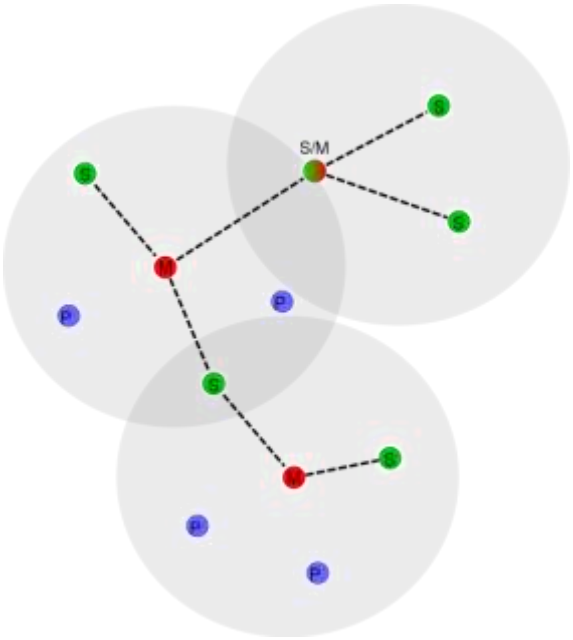
2.1.1 Bluetooth и Bluetooth Low Energy

Bluetooth е технологичен стандарт за безжичен обмен на данни на малки разстояния. Създаден е от Ericsson през 1994 година, като в началото е предназначен за безжична алтернатива на кабелния стандарт RS-232. Чрез Bluetooth могат да бъдат свързани няколко устройства, избягвайки проблеми със синхронизацията. Стандарта е поддържах от Bluetooth Special Interest Group (SIG). По-рано Bluetooth е бил стандартизиран от IEEE като IEEE 802.15.1, но в момента спецификацията се поддържа и доработва изцяло от Bluetooth SIG [21].

Bluetooth Low Energy е въведен със стандарта Bluetooth v4.0 (наричан също Bluetooth Smart) и е в експлоатация от 30 Юни 2010 година. Bluetooth Low Energy е базиран на технология известна като Wibree, разработена от Nokia през 2006 година и има за цел драстично да понижи консумацията на ток в мобилни устройства [21]. Стандарта Bluetooth позволява безжично свързване на устройства и обмен на данни помежду им. Връзката се базира на принципа master и slave, като по стандарт едно master Bluetooth устройство може да има максимум 7 прикачени slave Bluetooth устройства към него. В Bluetooth Low Energy се използват Client и Server терминологиите, като там няма специфицирано ограничение по броя устройства - това зависи от ограниченията в конкретната имплементация на системата. Устройства, осъществили връзка помежду си се намират в така наречената piconet [22] (ad-hoc компютърна мрежа, използваща технологията Bluetooth) и могат да разменят ролята си по време на осъществена връзка, стига да синхронизират това решение помежду си (това се отнася само за две устройства спрямо едно-друго). За пример може да се даде следната ситуация - от начало Bluetooth слушалки се включват като master за да започнат връзка с мобилно устройство, но в последствие след като връзката е осъществена, ролите могат да бъдат разменени. Bluetooth спецификацията позволява свързването на две или повече piconet-и да сформират

scatternet - това е мрежа, в която няколко устройства едновременно играят ролята на master устройства в една piconet и slave ролята в друга piconet [21].

На Фигура 4 може да бъде видяна scatternet, като зеления цвят са slave устройства, червения цвят са master устройства, а синия цвят са “паркирани” устройства [23].



Фигура 4 [<wikiscatternet7>]

Във всеки един момент могат да бъдат обменени данни между master-а и някое друго устройство (освен, ако другото устройство не е само broadcaster). Master устройството избира кое slave устройство да бъде адресирано. Обикновено master устройството сменя адресираното устройство много бързо, когато има няколко такива свързани към него, ползвайки така наречения Round-robin scheduling (система за разпределение на задачите, където всяка задача си има определен времеви слот) [21].

2.1.1.1 Начин на работа на Bluetooth

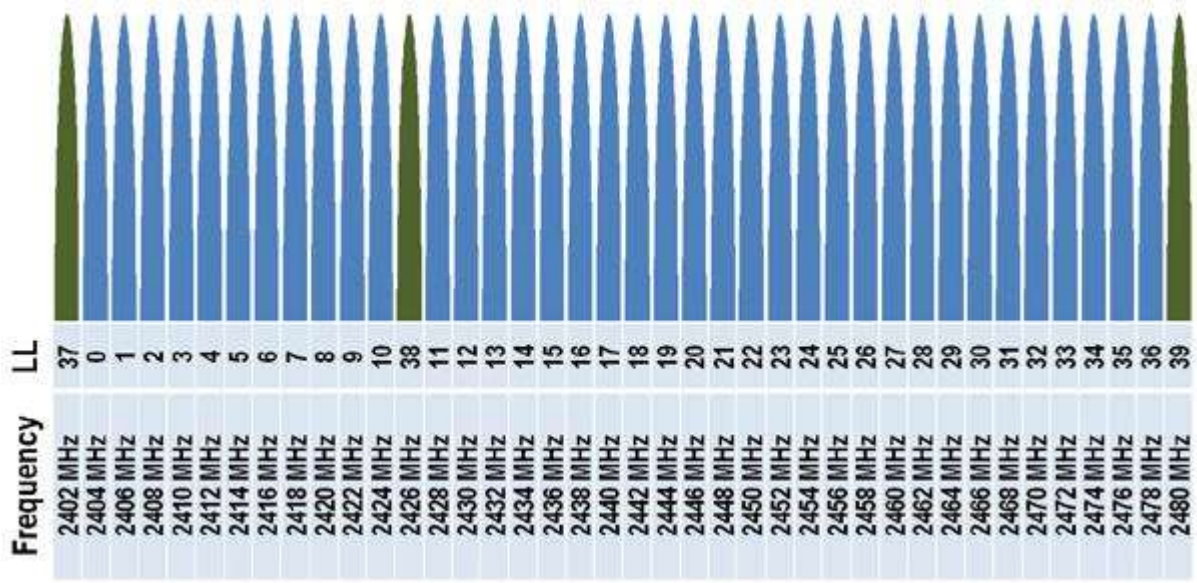
Bluetooth използва честотите между 2400 и 2485 MHz [24], което се намира в глобалната ISM късо-обхватна честотна мрежа, като има за цел и ниска консумация на енергия. Обхвата на устройството зависи от мощността на сигнала, който излъчва. В този аспект Bluetooth е разделен на 3 класа устройства, които могат да бъдат видени в Таблица 2.

Клас	Максимална позволена мощност		Обхват (метри)
	(mW)	(dBm)	
1	100	20	~100
2	2.5	4	~10
3	1	0	~1

Таблица 2 [21]

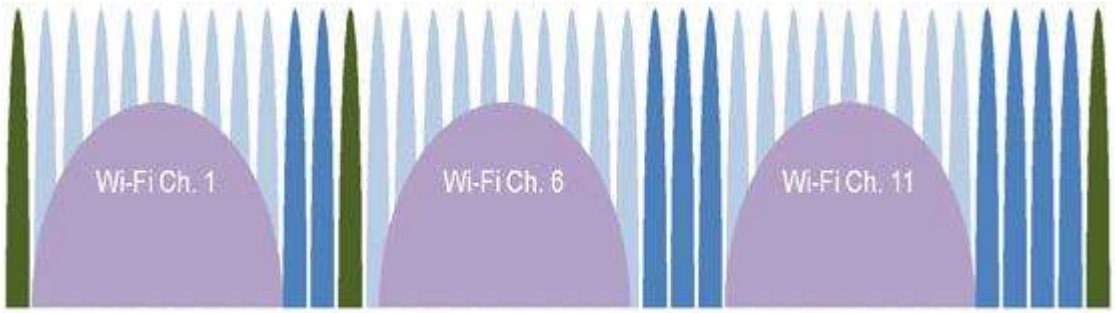
Технологията ползва Frequency-Hopping, като Bluetooth Low Energy устройствата предават пакети с информация в един от 40 канала с 1 MHz честотна лента (каналите са разположени през 2 MHz, като първия канал започва на 2402 MHz, а последния свършва на 2480 MHz), като се поддържа и Adaptive Frequency-Hopping. Ползвайки Adaptive Frequency-Hopping обикновено се извършват 1600 скачания на канали [<wikibluetooth2>].

Наличните канали за Bluetooth Low Energy могат да бъдат видени на Фигура 5. Трите тъмно зелени канала, 37-ми, 38-ми и 39-ти са така наречените advertising канали (канали, в които дадено устройство се "рекламира" и може да бъде засечено от останалите устройства, които сканират за да осъществят връзка с него), а останалите са канали за данни (data channels) [25].



Фигура 5 [25]

За по-надеждна комуникация се ползва и Adaptive Frequency-Hopping. Той надгражда върху обикновения Frequency-Hopping, като избягва изцяло канали, в които има смущения (излъчвания от други устройства, умишлени смущавания на каналите и др.), както може да бъде видяно на Фигура 6 [<digikeyble6>].



Фигура 6 [25]

Фигура 6 илюстрира как канали, в които има смущения от Wi-Fi (Wi-Fi също използва ISM честотната лента) сигнал не биват ползвани от Bluetooth.

a) Bluetooth устройства

За предаване на информация, сигнала на Bluetooth устройствата е дигитално модулиран.

Всяко Bluetooth устройство може да поддържа дадени услуги (services)/профили. Например дадено устройство може да имплементира профила за Hands-Free. Тези профили имат идентификатори, наречени Universally Unique Identifier UUID, което може да е от 16 до 128 бита. 16 битовите версии трябва да бъдат регистрирани в списък, поддържан от Bluetooth SIG и съответно за тях се плаща лицензна такса, докато 128 битовите профили са за свободно ползване, тъй като шанса два профила да са еднакви е много малък.

b) Bluetooth Low Energy

Bluetooth Low Energy (също известно като Bluetooth Smart) ползва същите честоти и механизми като класическия Bluetooth, но има различен протокол за предаване на информация. Той е предимно предназначен за мобилни устройства, които могат да бъдат носени върху човек (така наречените wearable devices) и биват захранвани от

батерии. Именно за това е нужна ниската консумация на енергия, която се осъществява благодарение на Bluetooth Low Energy стандарта.

Устройствата ползващи Bluetooth Low Energy стандарта обменят данни не чрез потоци от информация, а чрез парчета информация. Това означава, че концепцията не е например обмен на файлове, а обмен на състояния, стойности и подобни. По този начин има възможност да бъде спестена консумация на енергия, тъй като устройствата не изпращат постоянно данни, а само когато предаването на информация трябва да се предаде през даден интервал или бъде изискано от дадено устройство. Например устройства, могат да изпращат данни за текущия час, температура, скорост или др. при всяка промяна на стойността или на всяка секунда [<wikible5>].

Ако трябва обектно да се представят две Bluetooth Low Energy устройства, то това, което предоставя данни се нарича сървър (периферното устройство), а това, което чете данните се нарича клиент (централното устройство). Както вече е споменато, Bluetooth устройствата използват профили. В случая на Bluetooth Low Energy те се наричат и attributes (атрибути). Атрибутите биват представяни от сървъра (че той ги поддържа) и биват използвани от клиента. Всеки атрибут си има UUID, като те са с големина 16 бита, ако са стандартизирани от Bluetooth SIG и с дължина 128 бита, ако са измислени от производителя на Bluetooth устройството [26].

Bluetooth протокола поддържа няколко действия:

От клиента: Търсене/Намиране на други устройства, Четене на данни от други устройства, Писане на данни в други устройства, Потвърждение и Индикация

От сървъра: Отговаря на действията от страна на клиента, Изпращане на нотификации до клиента и Индикация [26].

- GAP и GATT

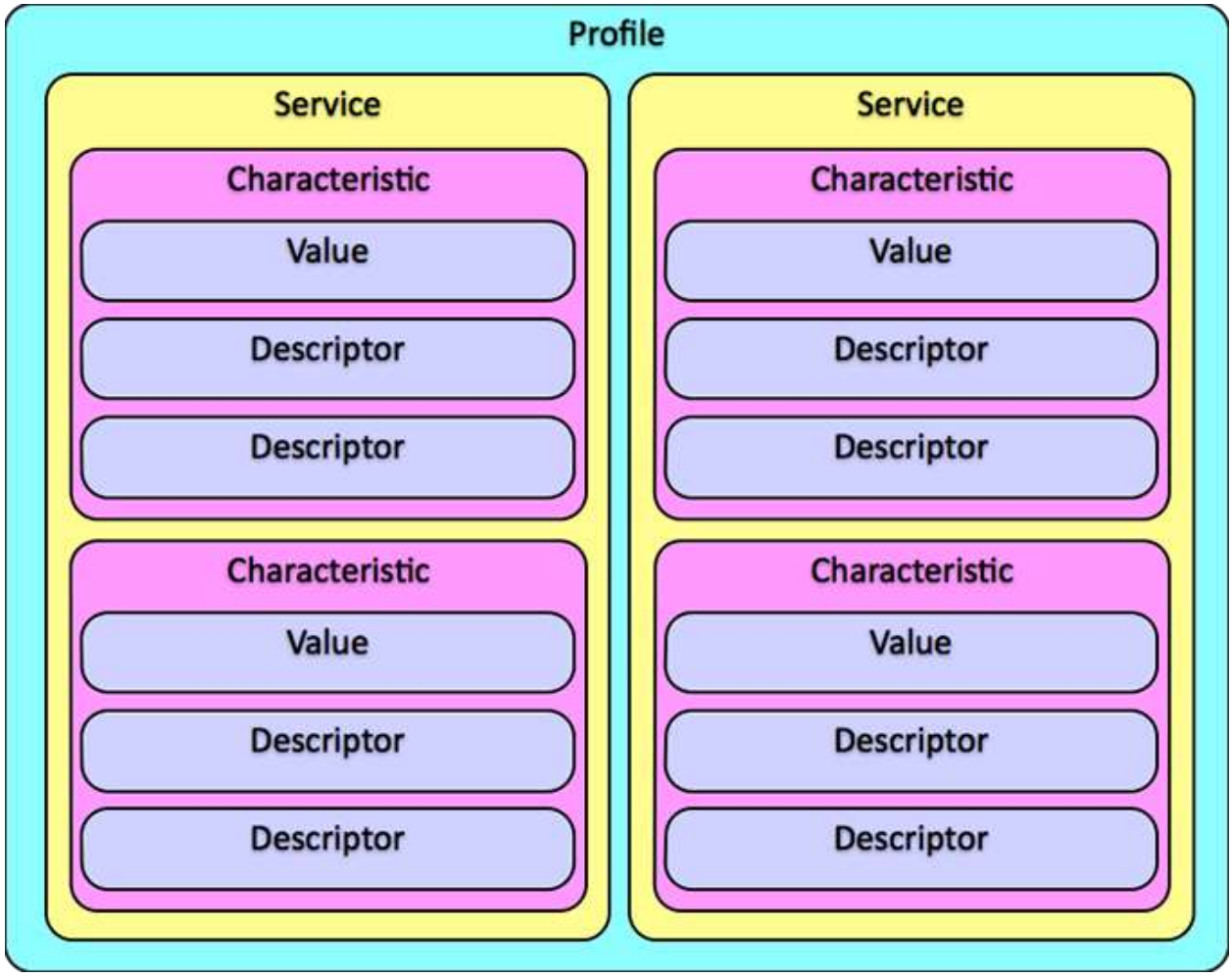
  GAP или Generic Access Profile съдържа най-общата информация за даденото Bluetooth Low Energy устройство и е задължително за всяко едно такова. В GAP се намира името на устройството, какво представлява то и предпочитани

параметри за осъществяване на връзка с него (скорост/интервал на връзката и др.). Предпочитани параметри означава, че устройството, което ще се свързва към него не е длъжно да уважи тези параметри и да ползва тях за да осъществи връзка, но просто това би било най-оптималното решение. GAP информацията е ключова при рекламирането на дадено устройство, тъй като именно тя бива излъчвана. [22]

GATT означава Generic Attribute Profile. Bluetooth Low Energy е изграден около идеологията за GATT, ползвайки го за структуриран подход към описването на това как периферните устройства показват данни на други устройства. Информацията за периферните устройства е организирана в колекции от Услуги (Services), които описват логическите функционалности на даденото устройство (например даден сензор, като термометър ще има собствена услуга). Всяка услуга съдържа колекция от характеристики (Characteristics), които се ползват за обмен на дискретни данни между устройствата (например характеристика може да бъдат данните от температурния сензор), като в услугите може да има и Дескриптори (Descriptors), които дават допълнително описание за дадената характеристика (например в какви мерни единици са предоставените стойности от температурния сензор на даденото устройство). [27]

Характеристиките обикновено съдържат флагове за свойствата им и стойност. Флаговете дават допълнителна информация за дадената характеристика, като например дали тя може да бъде прочетена или писана (readable or writable) и дали поддържа нотификации (например при промяна на стойността й да уведомява клиентското устройство за това събитие). [27]

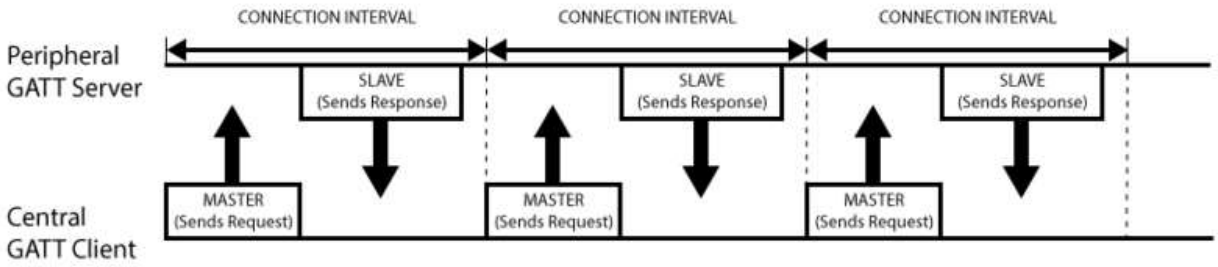
На Фигура 7 е илюстрирана структурата на GATT профила.



Фигура 7 [27]

Всичките елементи - услугите, характеристиките и дескрипторите се наричат атрибути и от там идва името на GATT. Всеки един атрибут се идентифицира с UUID.

След като е осъществена връзка между две устройства, всички транзакции на данни се стартират от клиента, като той получава съответните отговори от сървъра, освен когато не е активирана дадена нотификация и сървъра изпраща данни при промяната им (или някакво друго условие). Фигура 8 илюстрира типичната комуникация между клиент и сървър в Bluetooth Low Energy връзка. [28]



Фигура 8 [28]

В Таблица 3 може да бъде видяно директно сравнение между класическия Bluetooth стандарт и стандарта Bluetooth Low Energy.

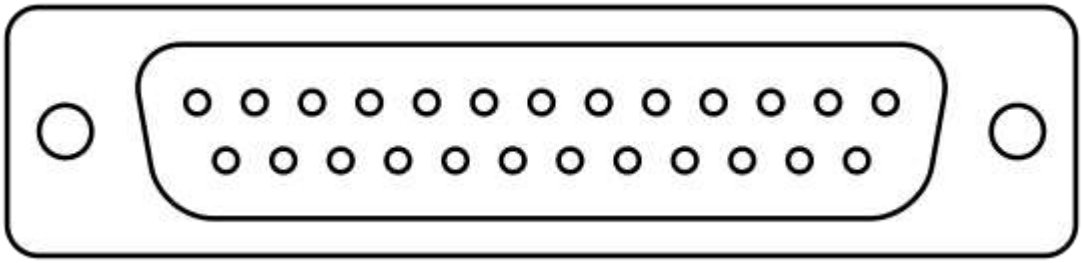
Техническа спецификация	**Класическа Bluetooth технология**	**Bluetooth Smart технология**
Дистанция/Обхват (теоритичен максимум.)	100 метра	>100 метра
Активни slave устройства	7	Не е дефинирано; Зависи от имплементацията
Техническа спецификация	**Класическа Bluetooth технология**	**Bluetooth Smart технология**
Сигурност	56/128-битова и потребителски дефинирана на приложния слой	128-bit AES с Counter Mode CBC-MAC и потребителски дефинирана на приложния слой
Устойчивост	Adaptive fast frequency hopping, FEC, fast ACK	Adaptive frequency hopping, Lazy Acknowledgement, 24-bit CRC, 32-bit Message Integrity Check

Латенция (от не-свързано състояние)	Обикновенно 100 ms	6 ms
Минималко общо време за изпращане на данни	100 ms	3 ms
Предаване на глас	Да	Не
Консумирана мощност	1 W as the reference	0.01 до 0.5 W
Пикова консумация на ток	<30 mA	<15 mA
Откриване на услуги	Да	Да
Концепция за профили	Да	Да

Таблица 3[29]

2.1.2 RS232

RS232 е телекомуникационен стандарт за предаване на данни по серийна комуникация. Популярен в миналото за комуникация между компютърни терминали и устройства/перфиерия, като модеми, мишки и др. Поради големината на конекторите и ниската скорост до голяма степен е изместен от USB стандарта, но все още се ползва в индустриални машини, мрежово оборудване и научни инструменти. На Фигура 9 може да бъде видян стандартен конектор за RS232 комуникация. [30]



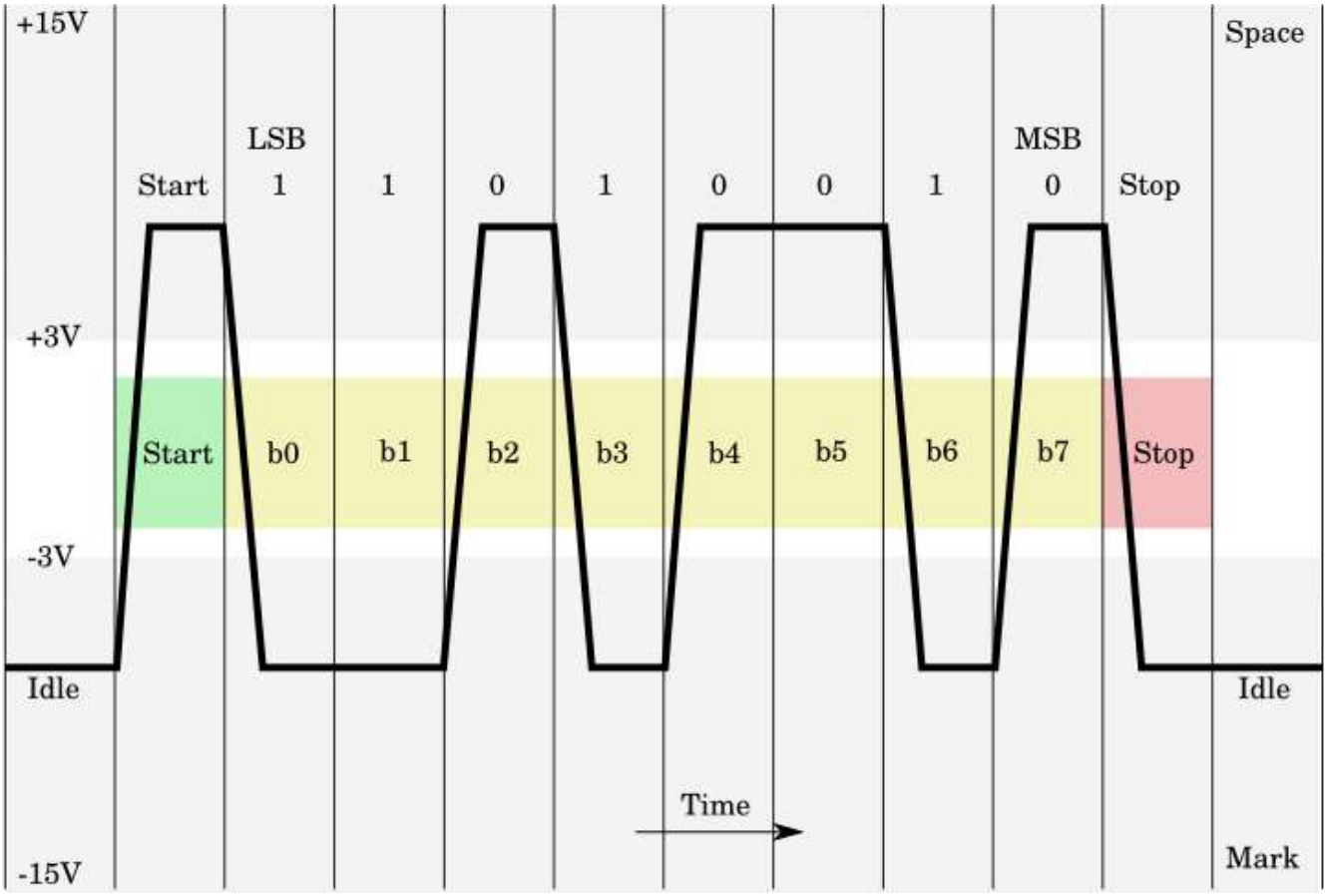
Фигура 9 [30]

Стандарта предава данните бит по бит в определени времеви слотове, определени от data-rate-а между устройствата. Стандарта дефинира няколко контролни схеми във всяко едно устройство, които да менажират връзката между терминала и приемащото устройство. Тъй като схемите за приемане и предаване на данни във всяко едно устройство са различни, това позволява протокола да поддържа full-duplex комуникация (дадено устройство да приема докато предава данни и обратното). [30]

Стандарта дефинира нива на напрежение, които са валидни при обмена на данни, които отговарят на логически нула или единица по мрежата за комуникация. Валидните сигнали са или в обхвата от +3 до +15 волта или в обхвата от -3 до -15 волта. Обхвата от -3 до +3 волта не се счита за валиден сигнал. При обмена на данни по линиите (TxD и RxD – съответно линия за предаване и за приемане) логическата нула е положително напрежение, а логическата единица е отрицателно напрежение спрямо общата маса. Важно е да се отбележи, че RS232 комуникацията може да става по кабел дълъг до 15метра, понеже по-дълги кабели биха имали прекалено голям капацитет. Специални кабели с нисък капацитет могат да бъдат ползвани за да удължат комуникацията до 300 метра, но отвъд тази дължина трябва да бъдат ползвани други протоколи за комуникация. [30]

Понеже в повечето случаи логическите нива на сигналите са по-големи от тези, на които работят схемите, които ги ползват, то често се ползват отделни драйвери/карти специално предназначени за RS 232 комуникация. [30]

На Фигура 10 може да бъде видяно предаването на буквата “K” в ASCII формат. Битовете включват един стартов бит, осем бита за данни и един стоп бит. Това е типичен формат за старт-стоп комуникация, но самия стандарт не дефинира формата на предаваните данни или подреждането на битовете. [30]



Фигура 10 [30]

В предложения набор от технологии за магистърската теза, комуникационния стандарт RS232 се използва като евтин и базов начин за дебъгване и подаване на команди към микроконтролера на системата. Недостатъка му е, че до платката трябва да бъде прокаран кабел от/към компютърно устройство.

2.1.3 SD Card

Примерната система поддържа монтиран четец и използва SD карти за запаметяване на логове или четене на различни конфигурационни данни, ползвани при функционирането й – записва изминатия път, чете зададени координати за навигиране и др. Конфигурацията през SD картата се ползва като най-ниско

приоритетна и се чете само ако не е осъществена друг вид комуникационна връзка към автомобила (Bluetooth, RS-232).

2.2 Измервателни

2.2.1 Намиране на текущото местоположение - GPS

Примерната система използва GPS приемник Quectel L80 за определяне на местонахождението. Модула L80 е избран, защото има ниска консумация в standby режим (1mA) и 20-25mA консумация по време на приемане на сигнали и следене на местоположението. [31]

Приемането на координати е изцяло автоматизирано и имплементирано в самия модул, като той периодично предава координатите по UART към микроконтролера. Скоростта за обновяване и скоростта на връзката може да се конфигурират посредством команди, изпратени по UART от микроконтролера към GPS модула, като настройките по подразбиране са обновяване на данните със скорост 1Hz и baud rate на UART 9600 bps. Конфигурирането позволява скоростта на обновяване да е от 1Hz до 10Hz, a baud rate да бъде от 4800bps до 115200bps. [31]

Максималното време за получаване на валидни GPS координати е около 35 секунди, като точността в идеални условия (благоприятни метеорологични условия и местоположение извън населени места с множество сгради) е 2.5 метра, като може да се ползва вградената антена (залепена от горната част на модула) или да се свърже външна такава. [31]

Физически GPS модула може да работи при температуро от -40 C до +85 C и може да издържа максимално ускорение от 4G. [31]

2.2.2 Засичане на препятствия - Ултразвуково отчитане

За ултразвуково безкондактко отчитане на дистанция примерната система използва сеознора HC-SR04. По спецификация отчита разстояние от 2 до 400 сантиметра и има точност до 3 милиметра. Работи на 5V напрежение и при работа консимира 15mA.[32]

За да функционира модула изпраща 40kHz сигнал и чака връщането му. Времето за което се върне е разстоянието до обекта пред него. [32]

Ето как изглежда и самият ултразвуков сензор:

Фигура 11

[32]

2.2.3 Ориентация в пространството - Компас

Примерната система използва електронен компас. Моделът е закрепен на готова платка за разработване - MOD-HMC5883L на Olimex.

Фигура 12

[33]

Самия електронен компас е модел HMC5883L и това са някои от главните му характеристики [<olimexcompass>]:

- Точност от 1 до 2 градуса
- Работно напрежение от 2.16 до до 3.6 волта
- Може да дава данни със скорост 160Hz (160 пъти в минута)

2.3 Управление

2.3.1 Микроконтролер

Микроконтролера ще управлява цялата примерна система и ще съдържа софтуера, който е специално написан за целта. Избрана е готова развойна платка STM32F4 Discovery Board, която има микроконтролер на ST, модел STM32F407VG със следните характеристики:

Ядро 32-bit ARM Cortex-M4F; 1MB flash; 192KB RAM.

Микроконтролера поддържа CAN и LIN комуникация, ако е нужно системата да имплементира CAN комуникация между различни устройства.

Примерната система ползва UART и SPI Комуникация, като микроконтролера има хардуерни драйвери за тях.

Развойната платка включва вграден debugger, поддържа захранване за външна периферия на 3.3 и 5 волта. За прототипиране е избрана тя, защото включва всичката нужна периферия за лесно и бързо изграждане на прототипен блок за

контрол на електрическата автономна кола. Също така процесора поддържа и операции с floating point, което прави изчисленията на координати към навигационните цели по-бързи. По този начин може да се ползва само един процесор. Като пример, ако се ползва по-бавен процесор, ще се наложи слагането и на втори такъв – един да изчислява координатите, другия да направлява колата, управлявайки електрическите мотори и следейки изминатото разстояние. Решението с един процесор (ако е възможно) винаги е по-изгодно от това с два или повече, понеже се избягва проблема с тяхната синхронизация и различно време на стартиране и преминаване в оперативен режим.

3 Сигурност

3.1 Практики за сигурно и надеждно програмиране

Някои от практиките, които се следват са описани в стандарта MISRA C, който се прилага в проектите на фирмите, пишещи софтуер за Automotive бизнеса. Този стандарт стъпва върху предишни допускани грешки в индустрията и бива създаден от опитни специалисти в сферата, като дефинира препоръки и забранява операции в написания код, които могат да доведат до не поискани странични ефекти и неконтролируемо поведение на системата, как трябва да бъде тестван кода и какви стандарти трябва да бъдат внедрени в предприятието за да отговаря на MISRA C. Самия стандарт е описан като “Guidelines for the use of the C language in critical systems”. [34]

По-долу са описани някои от важните (но не всички) практики, които е задължително да бъдат стриктно спазвани при написването на софтуер за системата:

- Като едно от първите и най-важни правила е нужно да не се допуска недефинирано и неспецифирано поведение на софтуера. Това означава, че всеки който пише код трябва да знае какво пише и да е наясно със страничните ефекти. За минимизирането на странични ефекти се правят код-ревюта на кода, както и изпълнение на множество тестове. [34]

- На всички автоматични променливи се присвоява инициализираща стойност, преди те да бъдат използвани. [34]
- В енумератори знака „=“ се ползва само за задаване на стойността на първата енумерираща константа. [34]
- Не са правени имплицитни конвертирания от един тип към друг. Където това е нужно се прави експлицитно, за да се види, че е направено с цел, а не по погрешка. [34]
- Конвертирането (кастването) от един тип към друг става само, когато новият тип е със същия знак като оригиналния (например signed char към signed int). [34]
- Добавя се suffix “u” на всички константи, които са от unsigned тип. [34]
- Не се ползват bitwise оператори върху променливи, които са от тип signed. [34]
- В условията за for цикъл не се ползват променливи от тип с плуваща запетая. [34]
- Променливите, които се ползва в условието за for цикъл не биват променяни в тялото на for цикъла. Те биват променяни единствено в условието на for цикъла. [34]
- Да няма код, до който не може да бъде стигнато по никакъв начин по време на изпълнение на програмата. [34]
- Да не се използва goto. [34]
- Функциите трябва да имат само един възможен изход от тях и той да се намира на края на функциите. [34]
- Всичките if, след което else-if изрази задължително трябва да свършват със else израз. [34]
- break; трябва да бъде слаган в края на всеки switch case, който не е празен (не се позволява изпадане в следващия case). [34]

- Функциите не трябва да бъдат дефинирани с променлив списък на входните аргументи. [34]
- Функциите не трябва да извикват самите себе си, нито директно, нито индиректно. Т.е. не се позволяват рекурсивните функции, тъй като не може лесно да бъде изчислено и гарантирано колко пъти ще се само-извикат и какъв ефект ще има това върху стека в паметта (може да се получи препълване на стека). [34]
- Идентификаторите, които са дадени при декларацията и дефиницията на дадена функция трябва да бъдат еднакви. [34]
- Функции, които не приемат параметри трябва да бъдат дефинирани единствено с void параметър. [34]
- Всички изходи от дадена не-void функция трябва да имат експлицитен return израз. [34]
- Аритметика с пойнтери трябва да се изпълнява само с пойнтери, които сочат към масиви или елементи от масиви. [34]
- Операторите за сравнение не трябва да бъдат ползвани за сравнение на пойнтери, освен в случаите когато се ползват за пойнтери към един и същ масив. [34]
- Декларациите на обекти не трябва да съдържат повече от 2 нива на индиректност чрез пойнтери. [34]
- Трябва да бъде ползвана защита срещу многократното включване (include) на файлове (такава защита най-често бива постигана чрез използването на #ifdef или #ifndef). [34]
- Не се използва заделяне на динамична памет (заделяне на памет в heap-а на системата). [34]

3.2 Софтуерна сигурност

3.2.1 Комуникационна сигурност

За гарантиране на сигурността на примерната система трябва да бъде използван метод за размяна на ключове при осъществяване на комуникация със свързано устройство. Всяко устройство, което се свързва към автономната кола трябва да изпрати своето ID, както и ключ, с който да се криптира информацията. Автономния автомобил също праща ключ от своя страна, с който свързаното устройство може да криптира данните, изпратени към колата. Това са така наречените частни и публични ключове.

Трябва да бъде използван криптографическия стандарт RSA – всеки участник в комуникацията има частен и публичен ключ. Нека за пример използваме комуникация между участници А и Б, като всеки от тях има частен ключ и публичен ключ. Даден частен ключ може да декриптира съобщение, криптирано със съответстващия му публичния ключ (частния и публичния ключ на един участник са специално изчислена математическа двойка). Публичните ключове се ползват само за криптиране на съобщения и съобщение криптирано с публичния ключ не може да бъде декриптирано със самия него. По този начин, ако участник Б иска да предаде криптирано съобщение на А, то Б криптира съобщението с публичния ключ на А и го изпраща на А. След съобщението може да бъде декриптирано само със съответстващия частен ключ, който само А знае.

Хубаво е, когато се имплементира система за сигурност тя да ползва вече утвърдени стандарти (какъвто например е RSA), а не туко-що измислени алгоритми за сигурност. Това трябва да бъде така, понеже вече наложилите се стандарти за сигурност се ползват много разширено и са тествани в изобилие, така че възможността за пробив в тяхната сигурност е сведена почти до нула.

3.2.2 Цялостност на софтуера

При safety critical системите е много важно когато бъде открит проблем/бъг той бързо и удобно да бъде отстранен. Когато една система е произведена в голямо количество, обновяването на софтуера може да бъде много трудоемко и скъпо, ако системата не позволява автоматично обновяване на софтуера. За да се постигне това

обаче, трябва да бъде гарантирано, че новия софтуер, който ще бъде свален на системата е работещ и целостта му не е била нарушена (от злонамерено лице, което иска да инжектира собствен код в системата или просто при грешно предаване на данните – прекъсване на сигнала или друг вид инцидент) преди да се запише като действащ и стария, работещ такъв да бъде заменен. За да се гарантира цялостност на данните е нужно най-малко да бъдат изпълнени следните условия:

- Комуникацията за сваляне на софтуер също да бъде криптирана, както комуникацията за обмен на команди и други видове данни от/към желаното устройство.
- Устройството, което изпраща новия програмен код под формата на пакети трябва да праща пакет само тогава, когато устройството, което ги приема му поиска такъв (в рамките на конфигурируемо максимално време – ако то изтече, комуникацията трябва да бъде прекратена, тъй като връзката се счита за прекъсната от потребител или поради технически причини).
- Всеки предаден пакет данни трябва да си има чексума, която да бъде пресмятана дали е правилна от устройството, което е получило данните. Ако това не е така, устройството получило данните трябва да поиска пакета на ново и така няколко пъти, докато той не бъде доставен без грешки. Ако това не се случи до няколко опита (конфигурируем брой опити), то процедурата по сваляне на софтуера трябва да бъде прекратена, тъй като очевидно в момента на извършване на операцията има някакъв постоянен проблем.
- След свалянето на всички пакети от новия софтуер, приемащото устройство трябва да изчисли чексума/hash върху целия такъв. Чексумата/хеша в случая може да е от по-сложен вид, като например MD5, някой от видовете SHA-X, CRC или други по желание. Това е нужна стъпка, понеже функцията за чексума на всеки пакет най-често е проста такава и въпреки, че шанса е много малък, то е възможно проверката за цялостност на даден пакет да даде

резултат ОК, когато всъщност в пакета има грешни данни (грешно положителен отговор).

Презаписването на стария софтуер с новия такъв трябва да се извърши чак тогава, когато всичките проверки за целостта на данните преминат успешно, гарантирайки правилното функциониране на новия програмен код. Желателно е презаписването на софтуера да става, когато системата има налично аварийно захранване, така че ако основното бъде прекъснато, то процеса по презаписване да не спре и системата да не може да изпадне в така нареченото "bricked" състояние – състояние, при което може да бъде оправена, само ако ръчно й бъде записан програмен код чрез дебъгер или друг метод на флашване. Това може да се избегне, ако новия програмен код не презаписва стария такъв. В такъв случай микроконтролера само променя флаг кой от двата софтуера трябва да бъде използван – например след като новия бъде свален на устройството и проверките му за сигурност минат успешно, процедурата по ъпдейтване променя флаг SW_USED = 2 и при стартиране на микроконтролера, неговия bootloader чете променливата и скача на инструкцията, която е първа за новия софтуер. Ако в бъдеще се види, че новия софтуер има някакъв проблем, то бързо може да се премине към стария, който все още се намира на устройството променяйки само флага, който използва bootloader-a. Това може да стане дори и по безжичен път, ако дадения bootloader поддържа приемане на команди от различни видове периферия.

II. ГЛАВА ВТОРА: РАЗРАБОТКА НА МОДУЛИ ОТ МИНИМАЛНИЯ НАБОР НУЖНИ ЗА ФУНКЦИОНИРАНЕ НА СИСТЕМАТА

1 Flow Control на програмните модули

Като главен entry-point на софтуера и управляващ модул е създаден прост scheduler на задачи/таскове, който изпълнява дадените програмни модули на предварително дефинирани интервали. Времевите интервали са твърдо дефинирани с цел издръжливост на системата и константно поведение – важни практики и цели, които трябва да се спазват във вградените (embedded) системи, за да бъде гарантирана

тяхната устойчивост и стабилност. Scheduler-а е вграден в безкраен цикъл, намиращ се във main функцията на програмата.

Изминалото време се акумулира в броячи, които се увеличава с единица на всяка милисекунда – това става благодарение на прекъсване от Timer2 брояча на микроконтролера. Броячите са от тип Unsigned Long, което означава, че е гарантирано минимум 32 битова променлива (с цел портативност на софтуера). Това му позволява да бъдат акумулирани 4,294,967,295 милисекунди преди да се получи препълване (overflow) на променливата – т.е. системата е предназначена да стои включена до 49.7 дни, след което потенциално биха възникнали проблеми в софтуерните модули, които следят изминалото време. Броячите са няколко и всеки един от тях е поставен в RAM паметта, като статуса им не се съхранява при изключване на устройството или спиране на захранването. Всеки един от тях си има и собствена цел:

- Системен брояч – брои изминалите милисекунди от както устройството е било стартирано. Използва се от различни компоненти в софтуера за измерване на изминало време от Х събитие или за изпълнение на периодични функционалности.
- Non-serviced ticks брояч – Използва се само от sheduler-а. Целта на този брояч е scheduler-а да се вика на всяка една милисекунда – в главния безкраен цикъл се прави проверка дали този брояч е по-голям или равен на единица и ако е се извиква scheduler-а и таймера се декрементира. Ако не е устройството минава в Low Power Mode режим за пестене на енергия и изчаква таймера да бъде инкрементиран (да минат една или повече милисекунди) за да активира наново Scheduler-а.
- RTC брояч – Това е брояч, който се ползва от Real Time Clock-а на устройството. Може да бъде сверяван чрез полученото време от GPS приемника (което е точно до 10ns) или през Android устройство чрез специалното приложение.

1.1 Комуникация

Микроконтролера има възможността, да комуникира с телефони ползващи Android и компютърни устройства чрез допълнителни хардуерни модули. Връзката с Android устройства се осъществява по безжичен път – посредством Bluetooth 4.0, а с компютърните устройства или чрез Bluetooth или чрез кабелна връзка RS232.

Комуникацията със всичките хардуерни модули (включително и GPS) се осъществява чрез пълнене на циклични буфери.

Цикличните буфери се състоят от масив от променливи тип char, пойнтер към началото на масива, пойнтер към края на масива, пойнтер към текущото положение за писане и пойнтер към текущото положение за четене от буфера. При попълването на буфера се инкрементира пойнтера за местоположение на писането в него, като се прави проверка дали е достигнат края на буфера (пойнтера за писане в буфера се сравнява с пойнтера за край на буфера). Ако края на буфера е достигнат, пойнтера за писане се приравнява на началния пойнтер на буфера. Също така след проверката за достигане на края на буфера се прави и проверка дали пойнтера за писане е настигнал пойнтера за четене. Ако е така, това би означавало, че апликацията не е обработила достатъчно бързо записаните данни в буфера и за да не се получи презаписване в такъв случай заявката за писане просто не се обработва. Всички тези операции биват извършвани в критична секция, понеже функцията за попълване на буфери се изпълнява в прекъсване от микроконтролера, а и също така за да се гарантира цялостта на данните – ако в текущото попълване на данните, микропроцесора е прочел променливите в някой от своите регистри и в същото време се получи прекъсване, което да ги промени в RAM паметта, то след това промените от второто прекъсване биха били заличени след като микропроцесора завърши операциите в регистрите си (със старите стойности) и ги запише в RAM паметта върху новите.

Четенето от цикличните буфери също става в критични секции за да се гарантира отново целостта на данните. Във функцията за четене първо се прави проверка дали

пойнтера за четене от цикличния буфер е равен на пойнтера за писане в него. Ако това е така – не се връщат данни, понеже няма нови такива. Ако пойнтера за писане не е равен на пойнтера за четене, то последния се увеличава с единица и се проверява дали е по-голям от крайния пойнтер на цикличния буфер. Ако е така пойнтера за четене се приравнява на началния пойнтер, Ако функцията стигне изпълнението до своя край (не е изпълнен return поради някаква грешка) се връща увеличеният с единица пойнтер за четене и се излиза от критичната секция.

Всеки един източник на данни (GPS, Bluetooth, Compass и др.) си има собствен цикличен буфер и прекъсване, което го обслужва. Именно в прекъсването на всеки един вид комуникация се пълни съответния цикличен буфер, като се коригира пойнтера, който сочи къде се намират последно записаните данни във дадения цикличен буфер.

1.2 Измервания

1.2.1 GPS

Микроконтролера получава GPS координати, благодарение на модула L80. Тези данни биват приемани по UART протокол, като при получаване на данни хардуерния модул за UART извиква прекъсване, където данните се записват в буфер за GPS. Когато буфера получи цяло GPS съобщение (приеме carriage return символ) слага GPS съобщението в string и го предава за парсване. Парсването конвертира получения GPS string в координати, които може да бъдат ползвани за изчисления, като също така извлича и информация за текущото време (която се използва и за сверяване на вътрешния Real Time Clock и калибриране на осцилатора). Парсването се осъществява символ по символ, като самия парсер очаква да има определени типове символи (букви, цифри) на определени места в string-а на GPS съобщението, като в същото време проверява и са валидността на данните (ако символите са правилни). При намиране на грешка текущото GPS съобщение се отхвърля и се изчаква сглобяването и получаването на ново такова. Този процес се изпълнява

постоянно (настройките по подразбиране на GPS модула L80 са да предава ново GPS съобщение на всяка една секунда).

Парсера може да чете два вида формати на GPS съобщенията - GPGGA и GPRMC. Данните, които се съдържат в тях, както и примерни съобщения са както следва.

- GPAA формат на съобщение, където всяка позиция в съобщението има своето предназначение:

  $GPGGA,hhmmss.ss,llll.ll,a,yyyyy.yy,a,x,xx,x.x,x.x,M,x.x,M,x.x,xxxx

  hhmmss.ss = Текущо време UTC

  llll.ll = Ширина

  a = N or S (Север или Юг)

  yyyyy.yy = Дължина

  a = E or W (Изток или Запад)

  x = Качество на GPS измерването (0=no fix, 1=GPS fix, 2=Dif. GPS fix)

  xx = Видими (използвани) сателити

  x.x = Смущение на хоризонталната позиция

  x.x = Надморска височина

  M = Мерна единица за височина

  x.x = Geoidal separation

  M = Мерна единица за geoidal separation, метри

  x.x = На колко време са данните от DGPS системата (секунди)

  xxxx = Идентификатор на DGPS станцията

	Примерни данни	Описание
Идентификатор на съобщението	$GPGGA	Global Positioning System Fix Data
Време	170834	17:08:34 Z
Ширина	4124.8963, N	41d 24.8963' N или 41d 24' 54" N
Дължина	08151.6838, W	81d 51.6838' W или 81d 51' 41" W
Качество на измерването: **- 0 = Invalid**	1	Качество на получените данни

- 1 = GPS fix - 2 = DGPS fix		
Брой сателити	05	Видими са 5 сателита
Horizontal Dilution of Precision (HDOP) – смущение в хоризонтална позиция	1.5	Относителна точност на хоризонталната позиция
Височина	280.2, M	280.2 метра над морското равнище
Height of geoid above WGS84 ellipsoid	-34.0, M	-34.0 метра
Time since last DGPS update	blank	Няма последно обновяване
Идентификатор на DGPS станцията	blank	Няма идентификатор на станцията
Checksum	*75	Ползва се от програми за да се засичат грешки при предаване на GPS съобщението

Таблица 4

[35]

- GPRMC формат на съобщение, където всяка позиция в съобщението има своето предназначение:

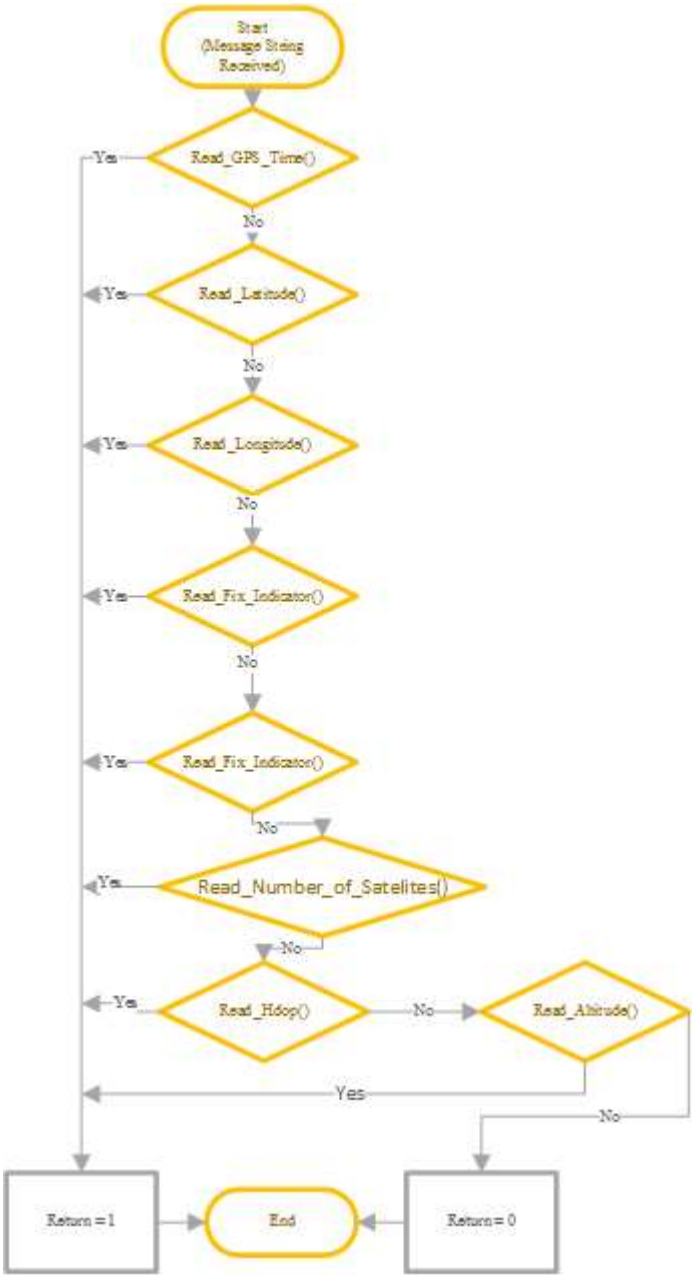
$GPRMC,220516,A,5133.82,N,00042.24,W,173.8,231.8,130694,004.2,W*70

	Примерни данни	Описание
Идентификатор на съобщението	$GPRMC	
Време	220516	
Валидност на сигнала	A	A = OK, V - невалиден
Ширина	5133.82	
Север/Юг	N	
Дължина	00042.24	
Изток/Запад	W	
Скорост	173.8	Скорост във възли
Курс	231.8	
Дата	130694	
Вариация	004.2	
Изток/Запад		
Checksum	*70	Ползва се от програми за да се засичат грешки при предаване на GPS съобщението

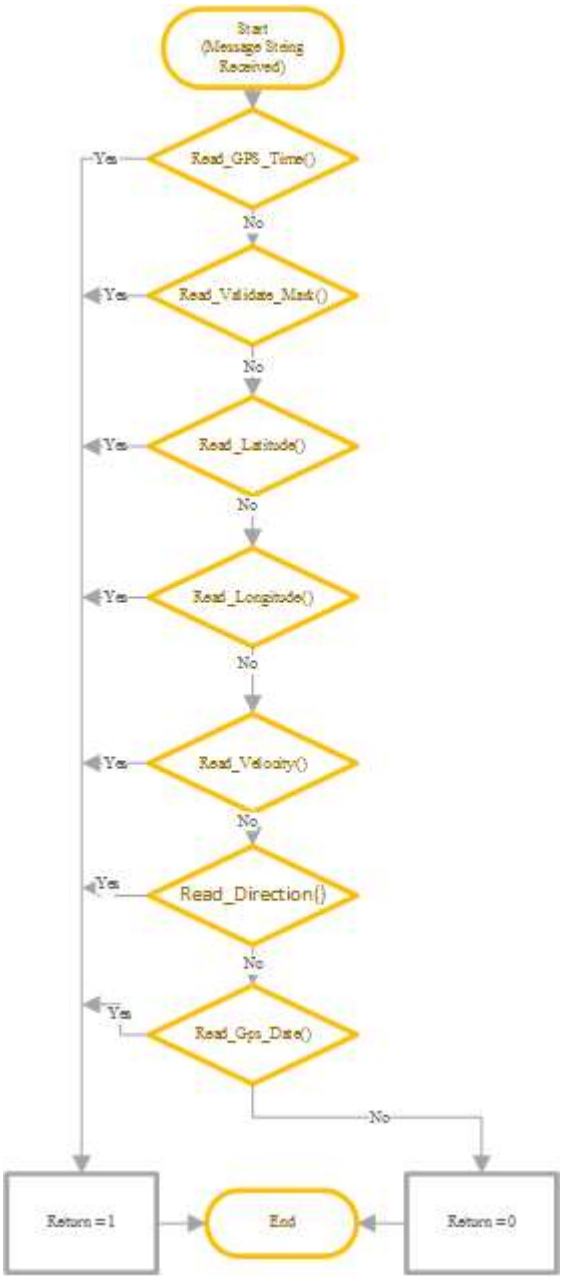
Таблица 5

[35]

Парсването на получените данни от GPS става чрез почти едни и същи функции, но викането им е с различна последователност, която зависи от това какъв формат съобщение е получено и трябва да бъде парснато. Това е така, защото двата вида съобщения съдържат в себе си почти еднакви данни, но с разменени позиции. Ето как изглеждат program flows на двата парсера:



Фигура 13



Фигура 14

На Фигура 13 е представено парсването на GPGGA формат на GPS съобщение, а на Фигура 14 е представено парсването на GPRMC формат на GPS съобщение. В условните фигури (ромбове) се викат функциите за четене на отделните данни, които се съдържат в GPS съобщенията, като в зависимост от това, дали четенето е успешно или неуспешно се взима решение как да се продължи. При връщане на 0 (No) – значи няма грешка и програмата продължава към четенето на следващите

данни. Ако всичките функции за парсване на GPS съобщение върнат резултат 0, то и главната функция за парсване, която ги вика също връща резултат 0 (че няма грешка) и попълва структурата с парснати GPS данни. При връщане на 1 (Yes) значи е имало проблем при четенето на данните и функцията за парсване връща резултат 1 и не обновява структурата, съдържаща парснатите GPS данни – остават си старите, които за последен път са били прочетени.

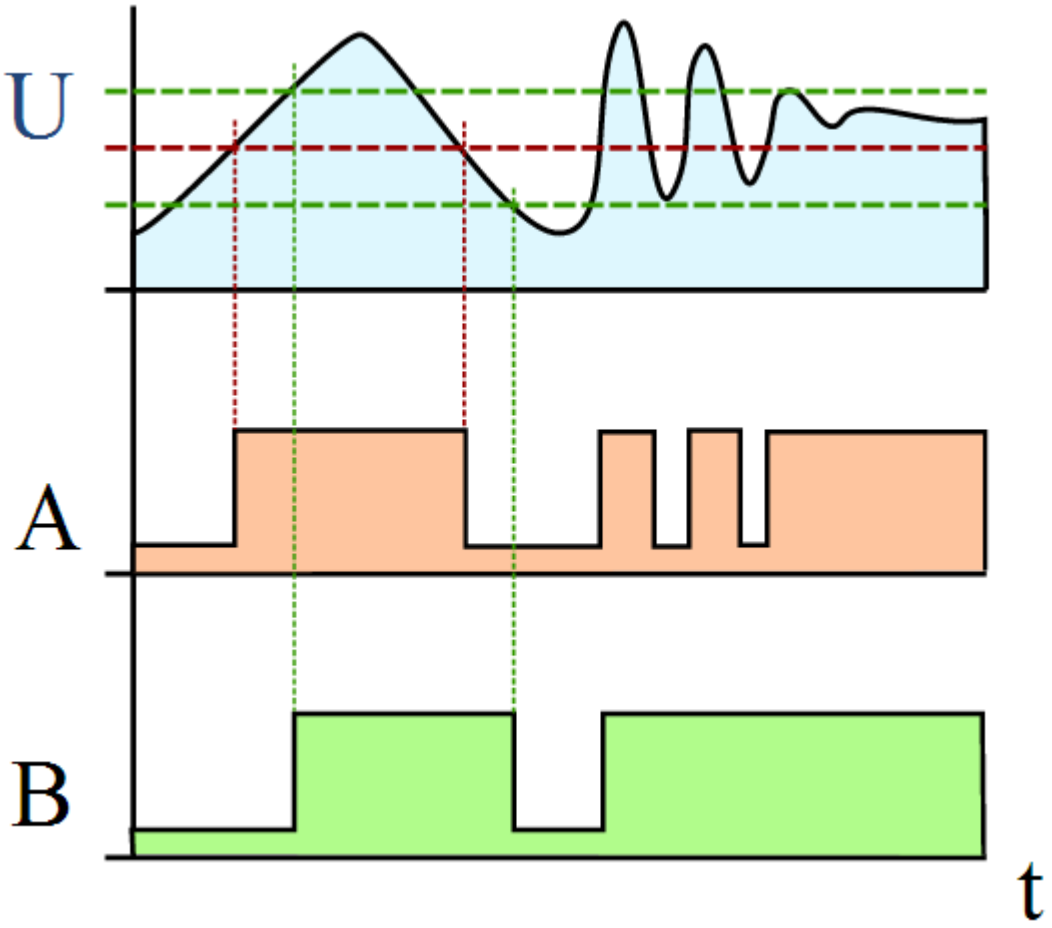
При успешно парсване на всяко съобщение (ако всички функции на парсера са върнали резултат 0 и главната функция на парсера успешно е попълнила структурата с парснати GPS данни), GPS парсера може да дава нотификации на модули, които имат нужда от новите данни – например модула за управление на Real Time Clock (сверяване на часа, ако полученото време от GPS и текущото системно време се различават над определена граница) и модула за изчисление на пътя, по който да се навигира автономната кола и калибриране на осцилатора (ако часовника постоянно изостава или избързва има възможност да бъде регулиран делителя на осцилатора, променяйки честотата, на която работи микроконтролера). Нотификациите представляват извикване на функции в абонирания за нотификация модул, където бива вдиган флаг. При изпълнение на функциите на дадения модул този флаг се чете и ако е вдигнат се предприемат съответните действия, като флага бива изчистван (нотификацията се нулира, за да може да бъде получена нова).

1.2.2 Напрежение

Софтуера постоянно следи напрежението от захранващата модула батерия. При спад на напрежението по определена граница – модула трябва да се изключи по начин, който гарантира безопасното спиране и уведомява потребителя. След това се чака напрежението да се вдигне до граница по-висока от тази, при която модула се е самоизключил т.е. има хистерезисно поведение. Хистерезиса е поставен, защото при включване на пълната функционалност на модула, той започва да консумира повече ток, от колкото докато е бил в режим на пестене на енергия и напрежението спада леко. Ако няма хистерезис ще се получи поведение, при което

когато напрежението е на дадена граница модула постоянно ще се превключва от състояние на пълна функционалност към състояние на пестене на енергия и обратно. Това би бил цикъл, повтарящ се докато напрежението на батерията не остане постоянно под границата за ниско напрежение (модула ще стои постоянно в режим за пестене на енергия). Имплементацията е аналогична с Schmitt trigger, което представлява:

Schmitt trigger – това е вид компаратор с хистерезис. Той превръща аналогов входен сигнал към дигитален изходен сигнал. При него има две граници на тригериране (границите представляват напрежение), които ще наречем граница 1 и граница 2, като граница 1 е по-голяма от граница 2. Когато напрежението на входа надмине граница 1, на изхода излиза дигитална единица (в случая на ползвания микроконтролер 5 волта). Когато напрежението падне под граница 2, на изхода излиза дигитална 0 (0 волта). Когато стойността на напрежението е между двете граници, тригера запазва състоянието на изхода (няма промяна). [36]



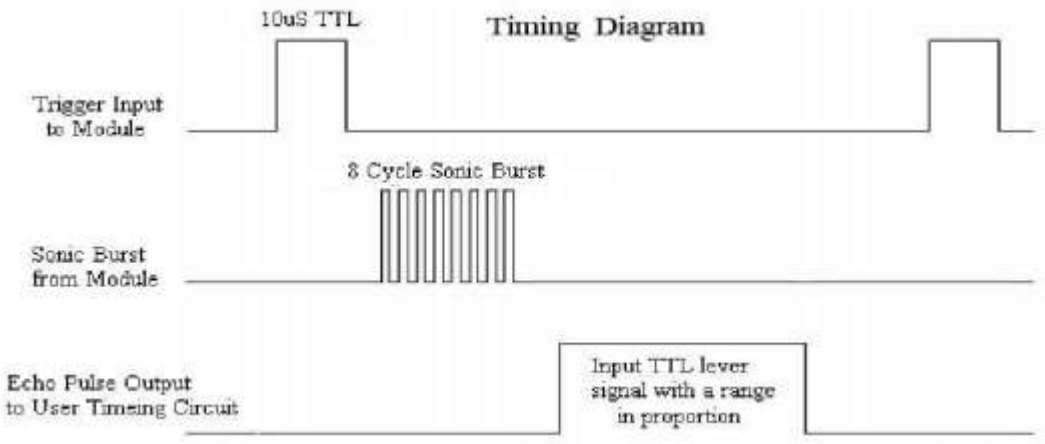
Фигура 15 [36]

(Изходен сигнал на Schmitt trigger (B) и на обикновен компаратор (A) при шумен (с много девиации, промени) вход (U))

От горната фигура може да бъде видяно как хистерезиса ползван в Schmitt тригера помага да се премахне шума от входния сигнал. Двете зелени черти на графиката при входния сигнал (U) са горната и долната граница на тригериране на Schmitt тригера – когато напрежението надскочи горната граница – изхода е 1, когато напрежението падне под долната граница – изхода е 0. Червената линия при входния сигнал е границата на тригериране на обикновен компаратор (понеже няма хистерезис, тя е само една). При него, когато напрежението е над границата – изхода е 1, а когато напрежението е под същата граница – изхода е 0. Може ясно да се види, че при обикновения компаратор има повече шум, от колкото при Schmitt тригера. [36]

1.2.3 Засичане на препятствия

За да бъде използван ултразвуковия сензор за разстояние е нужно да се следва определен набор от стъпки, които са описани в спецификацията му. Нужно е на един от пиновете му (Trig) да се подаде електронен импулс в рамките на 10uS, след което ултразвуковия сензор изпраща 8 40kHz-ови пулса и чака те да се върнат. След това да се изчака входен импулс от сензора към микроконтролера от пин Echo. Именно дължината на входния импулс определя дали е засечено препятствие и ако е, на колко разстояние се намира то. Следващата фигура илюстрира именно процеса на засичане на препятствие:



Фигура 16 [32]

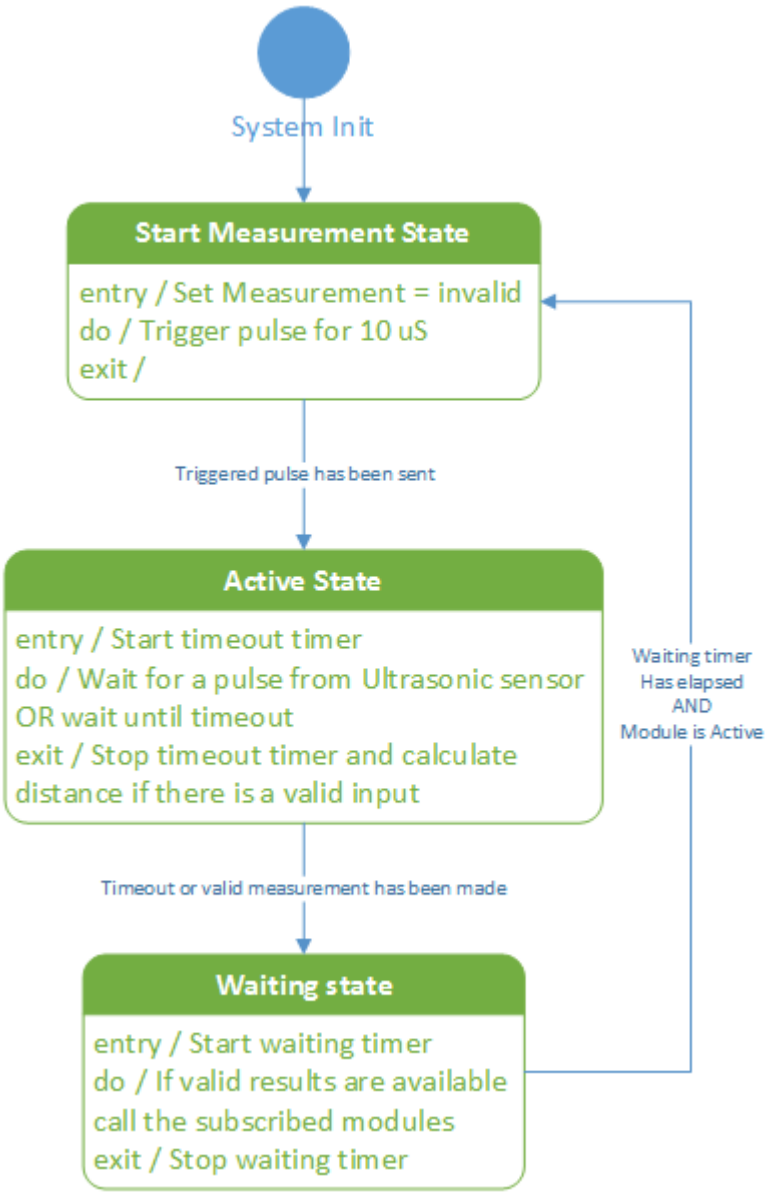
Самия модул има стейт машина за измерване – в първия стейт то се инициира, при което стейт машината се премества в състояние на изчакване. От състоянието на изчакване се излиза или докато не изтече таймер, който е с времетраене повече от времето, което ще отнеме на пулсациите от сензора да изминат 4 метра (максималното позволено измервано разстояние според спецификациите на сензора) или докато не бъде приет входния импулс от сензора с определена дължина. Когато едно от тези две условия се изпълни – ако има валиден резултат той се записва и разстоянието се изчислява, при което стейт машината минава в състояние на изчакване. Състоянието на изчакване е налично тъй като може да има конфигурирано време, което трябва да измине преди да започне ново измерване и чак след като то изтече, стейт машината преминава отново в първия стейт на

инициране на измерването. Междувременно при всяко извикване на главната функция на модула се проверява дали е има резултати от нови измерване и ако има валидни такива, то може да бъде извикана нотификация, както е обяснено по-долу. При засечени препятствие и определена далечина от него, модула за засичане на препятствия може да подава нотификации на други модули, които са регистрирани за тях. По този начин другите модули не е нужно постоянно да пуулват модула за засичане на препятствия, което прави системата по-оптимизирана. За постигане на целта, модула за засичане на препятствия съдържа таблица, в която се конфигурира при какво разстояние на засечени препятствие да се вика дадена функция от друг модул (подадена чрез пойнтер), която да служи като нотификация. Таблицата има следната структура (сложени са и примерни данни с илюстративна цел):

Дистанция до препятствието	**Нотификационна функция от абониран модул**
2m	NotifyObstacle_ModuleNavigation()
3m	NotifyObstacle_ModuleTrajectoryCalc()

Таблица 6

Стейт машината, извършваща измерванията може да бъде видяна на следващата фигура:



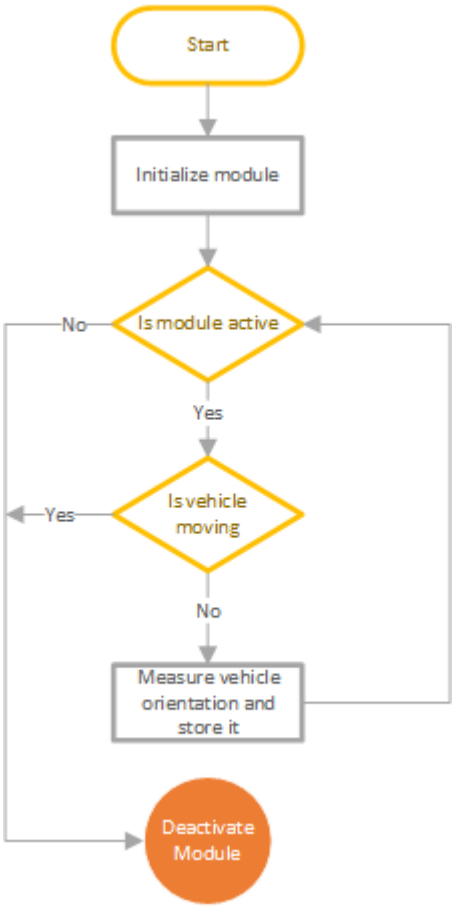
Фигура 17

Модула за измерване на разстоянието до дадено препятствие е важен в системата, но по желание може да бъде изключван – в такъв случай ако все пак таска му се вика, то стейт машината остава в състоянието за изчакване, тъй като не бива изпълнено условието за изход от него, където се изисква модула да е активен.

1.2.4 Компас

Модула за събиране и обработка на информацията от компаса е един от най-простите. Данните от компаса към микроконтролера идват в цифров вид, така че няма нужда от сложни схеми за изчисление и/или разкодиране на сигнала.

Единствената уловка е, че измервания чрез компаса трябва да се правят само при спрял автомобил, тъй като захранване на електродвигателите създава по-силни магнетични смущения, от колкото се усеща магнитното поле на земята и това кара дигиталния компас да дава грешни резултати. За това условието да се прочете компаса е докато колата е спряла и модула за компас е активен, след което може да бъде направено измерване и резултатите да бъдат запазени за бъдещата им обработка от други модули. Блок диаграмата на модула е както следва:



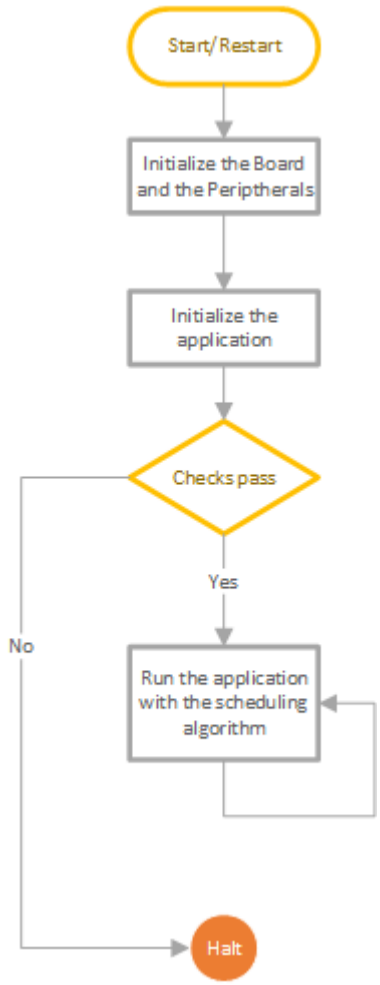
Фигура 18

Както се вижда на Фигура 18, модула за измерване на ориентацията се деактивира ако колата е в движение. Това се прави, понеже по време на движение не се събира информация за ориентацията и по този начин, чрез изключване на модула се спестява процесорно време.

1.3 Управление

Софтуера за управление е разделен на различни модули, които се грижат за определени задачи. Това модулно разделение на софтуера го прави лесен за разбиране и усъвършенстване/добавяне на нови функционалности.

Преди да започне изпълнението на каквито и да било задачи и таскове, първото нещо, което се прави от софтуера е да се инициализира и конфигурира всичката необходима периферия, която ще бъде ползвана на микроконтролера и извън него, както и да се инициализират всички данни, които ще ползва апликацията. Това се прави в main функцията преди влизането в главния безкраен цикъл след всяко стартиране или рестартиране на системата, като инициализирането на всички софтуерни модули представлява просто извикването на техните инициализиращи функции (съответно всеки модул, ако иска/трябва да бъде инициализиран трябва и да имплементира инициализираща функция) – това също помага за модулността на системата, като прави изключването и включването на модули по-гъвкаво и лесно. Следващата диаграма показва стъпките, които се изпълняват при инициализиране на цялата система:



Фигура 19

Веднага след инициализациите на периферията и апликацията се правят и проверки дали самата инициализация е преминала успешно и всички системи са в изправност – например за апликацията се проверяват масивите от данни, които са инициализирани, а от към периферията се пускат тестови съобщения към GPS Модула, на които той отговаря. При засечена грешка или неуспешна инициализация от която и да е страна, системата не продължава операция и бива спирана, като това се индикира със запалване на червен свето диод на главната платка. От това състояние може да бъде излезнато само чрез рестарт на системата, тъй като то представлява безкраен цикъл. Това е така, понеже когато изпълнението стигне до него не се викат повече функции и програмния стек остава непокътнат от апликацията, благодарение на което при дебъгване може да се видят какви грешки са довели до това грешно състояние на системата. Лесната диагностика води и до

лесно намиране на проблема и неговото отстраняване, което позволява бързото прототипиране и подкарване на системата.

След като успешно са конфигурирани всички нужни периферии и са инициализирани всички нужни данни, които ще ползва и апликацията управлението се предава на безкрайния цикъл, който в себе си съдържа различните си логически програмни модули, които позволяват системата да изпълнява функциите си. Модулите са разпределени както следва:

1.3.1 Модул за обработка на събраната информация от сензорите

Този модул има за цел да съберем информацията от сензорите и да я подготви в оптимален формат, ползван от модула за изчисление на траекторията и модула за навигация по изчислената траектория.

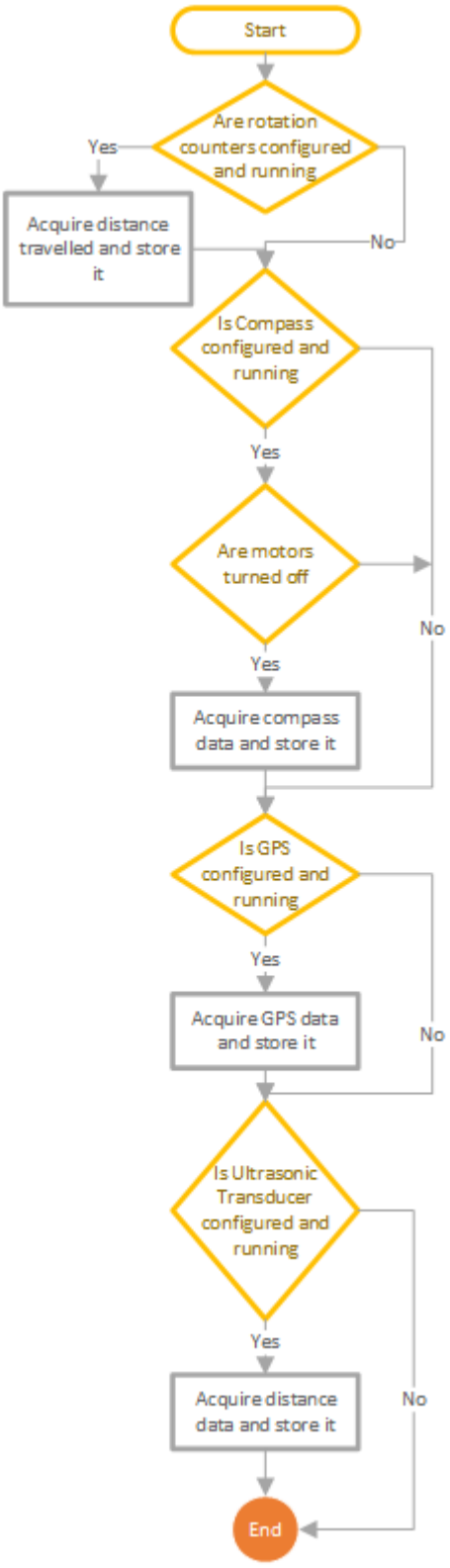
Събираната информация е от GPS сензора, ултразвуковия сензор за разстояние, сензорите за изминато разстояние и електронния компас. Събирането на информация от сензорите става само, когато те са включени, конфигурирани и е имало време да съберат нужните данни за тяхното безпроблемно функциониране. Проверката за тяхното състояние се прави всеки път преди да бъдат прочетени данните от тях с цел събиране на цялостни и точни данни. По този начин се гарантира правилното функциониране на навигационната система, тъй като в противен случай навигацията по траектория би довели до непредвидени обстоятелства.

Като пречка в текущия дизайн е важно да се отбележи, че тъй като електрониката не е екранирана измерванията от компаса се взимат в предвид само, когато колата е стационарна и не се движи. От експериментите при прототипиране става ясно, че компаса много лесно се смущава от по-силни токове в колата – например, когато бъде подавано напрежение към моторите за движение.

Модула може да събере информация от всички сензори или само от някои, като това зависи от конфигурацията и дали са включени. Събраните данни се ползват от други модули, като другите модули имат за задача да решат дали да ползват информация

от част от сензорите или е нужно всички сензори да са включени и събрали данни за да могат дадените модули да функционират. Например сверяването на часовника изисква само наличието на валиден GPS сигнал и не се интересува от информацията, предоставена от другите сензори, докато за успешно и безпрепятствено навигиране е нужна информация от сензорите за изминато разстояние, GPS, компасът, сензорите за препятствия.

На Фигура 20 може да бъде видян примерният алгоритъм на модула за събиране на информация от сензорите.



Фигура 20

По-долу е описана функционалността и на модулите, които изискват информация от модула за събиране на данни от сензорите.

1.3.2 Модул за изчисление на траекторията

Модулът за изчисление на траекторията взима като параметри текущото местоположение на устройството и желаната дестинация. Целта на модула е да изчисли директен път от сегашното местоположение до дестинацията, като модула не взима в предвид препятствия и труден терен по маршрута (тази допълнителна информация не бива взимана в предвид и не му бива подавана като входни параметри). Изчислената траектория се пази като двойка от команди – разстояние, което трябва да бъде изминато и с какъв завой трябва да бъде изминато то (ляв, десен или движение в права посока). Примерна изчислена траектория може да изглежда както следва:

Разстояние	Завиване
3m	Turn_Right
50m	Drive_Straight

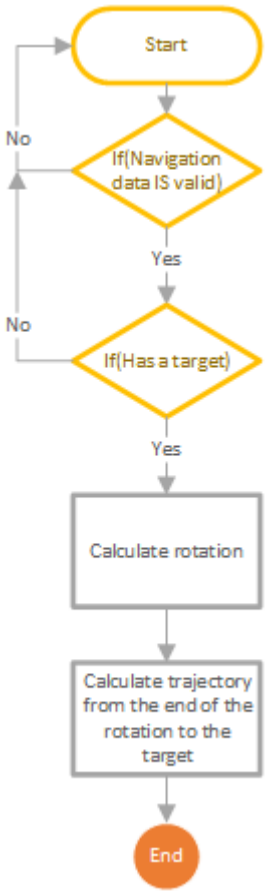
Таблица 7

Командите в Таблица 7 означават – завивай на дясно докато не изминеш 3 метра и след това карай направо докато не изминеш 50 метра. Тъй като колата може да завива изцяло надясно/наляво не се подава като параметър ъгъл на завой. Това е така, защото за завиване не се ползват стъпкови мотори, а обикновени безчеткови мотори.

Като статични параметри за автомобила модула разполага с информация за това колко е радиусът на завой на автомобила. Знаейки това и на базата на получената информация от компаса и GPS модула, модула за изчисление на траекторията има за първа задача да изчисли на колко градуса трябва да се завърти колата за да сочи към дестинацията. Всички изчисления стават благодарение на готови API-та от google maps – от там, чрез smart phone директно могат да бъдат взети разстоянието между текущото положение и цента, както и ъгъла на ориентация. Имайки изчислен ъгъла, на който трябва да бъде завъртян автомобила, модула изчислява колко

разстояние е нужно да бъде изминато с пълен завой (наляво или надясно) за да бъде постигнато желаното завъртане.

След като бъде изчислено завъртането и как трябва да бъде направено, алгоритъма трябва да изчисли пътя по права линия, който колата трябва да измине след като е направила завоя за да достигне правилна ориентация до целта си. Стъпките за изчисление на траекторията могат да бъдат обобщени във Фигура 21:



Фигура 21

API-тата на Google Maps дават дистанцията по повърхността на земята, а не по права линия от точка координати до точка координати (линия, която би минавала под земната повърхност).

След като са изчислени, маневрите се подреждат в таблица, както е показано в Таблица 7 и управлението на колата се предава на модула за навигиране по изчислената траектория.

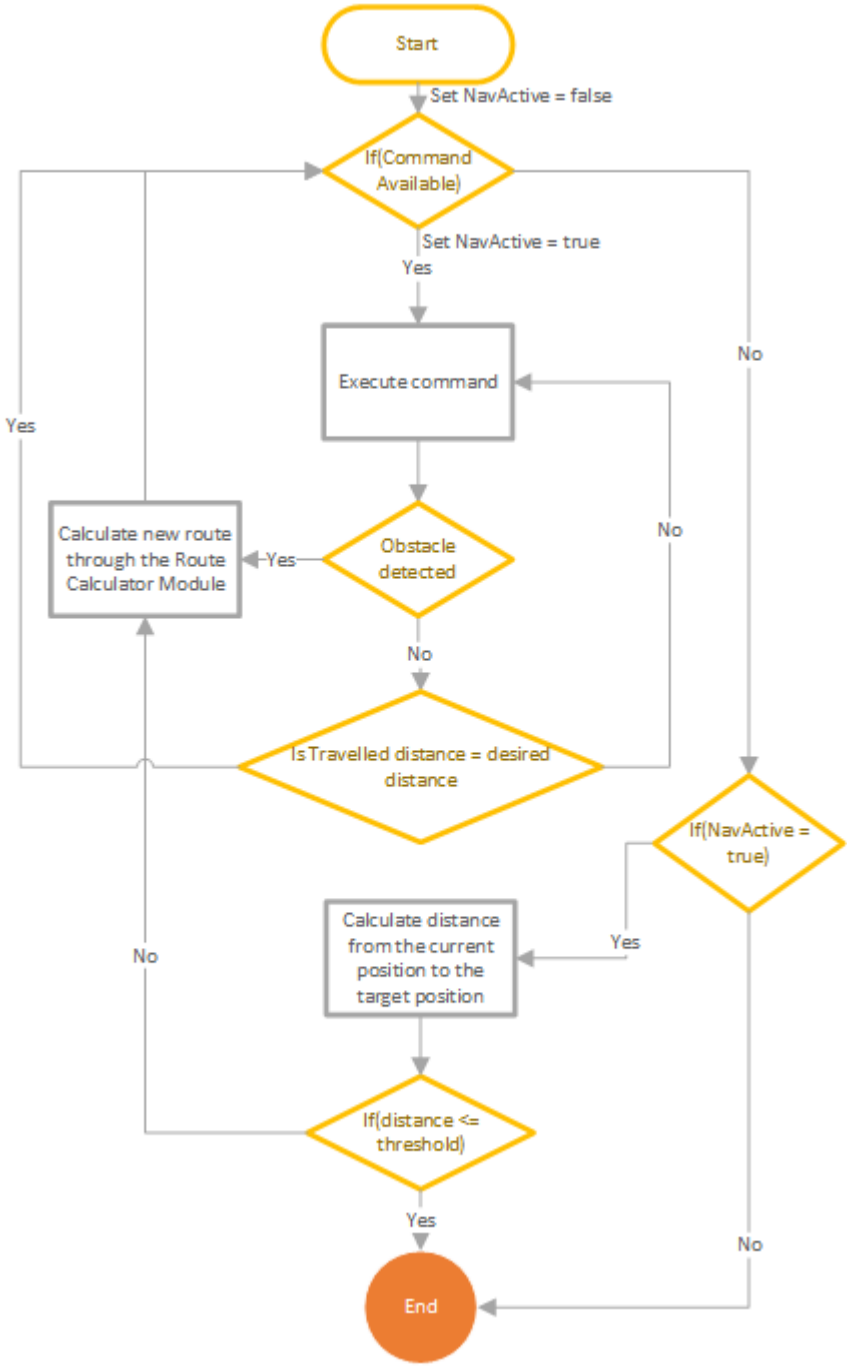
1.3.3 Модул за навигация по изчислената траектория

Модула за навигация по изчислена траектория взима като входни данни изчислената траектория от модула за изчисление, както и команда от потребителя за начало на навигацията. Този модул отговаря за следенето на изминалото разстояние и правилните команди за завиване, които трябва да бъдат изпълнени. Например модула е получил като изчислена траектория, че колата трябва да се движи три метра със десен завой. Модула дава команди към електрическите двигатели за ход напред/назад и за завой и държи тези команди, докато не изчисли, чрез сензорите за изминали разстояние, че са изминати 3 метра. След правилно изпълнение на тази команда, модула преминава към следващата в списъка и така докато командите не свършат.

Когато бъде изпълнена и последната команда модула прави запитване за текущото местоположение на колата. Ако то се разминава над определена граница с целевото положение, то бива извикван модула за изчисление на траектория от текущото положение и навигацията започва от начало докато не се достигне желаната дестинация в рамките на определена граница.

По време на движение, чрез ултразвуково сканиране може да бъде засечено, че има препятствие пред колата. Тогава колата спира и бива изчислен маршрут със два 90 градусови завоя в противоположна посока, първия от които е насочен в посоката, в която няма препятствие (ляво или дясно), като първо колата изминава назад няколко свои дължини за да може да завие безпрепятствено. Ако не може да се завие нито наляво, нито надясно, то колата се връща по пътя, който е изминала в изходящото си положение, като за целта се предават сегашните координати и началните като цел на модула за изчисление на траекторията. Ако по време на връщане към началната позиция също бъде засечено препятствие, то навигацията спира, тъй като се счита, че терена се променя и не може да бъде сигурно дали ще има възможен път обратно до желаната точка.

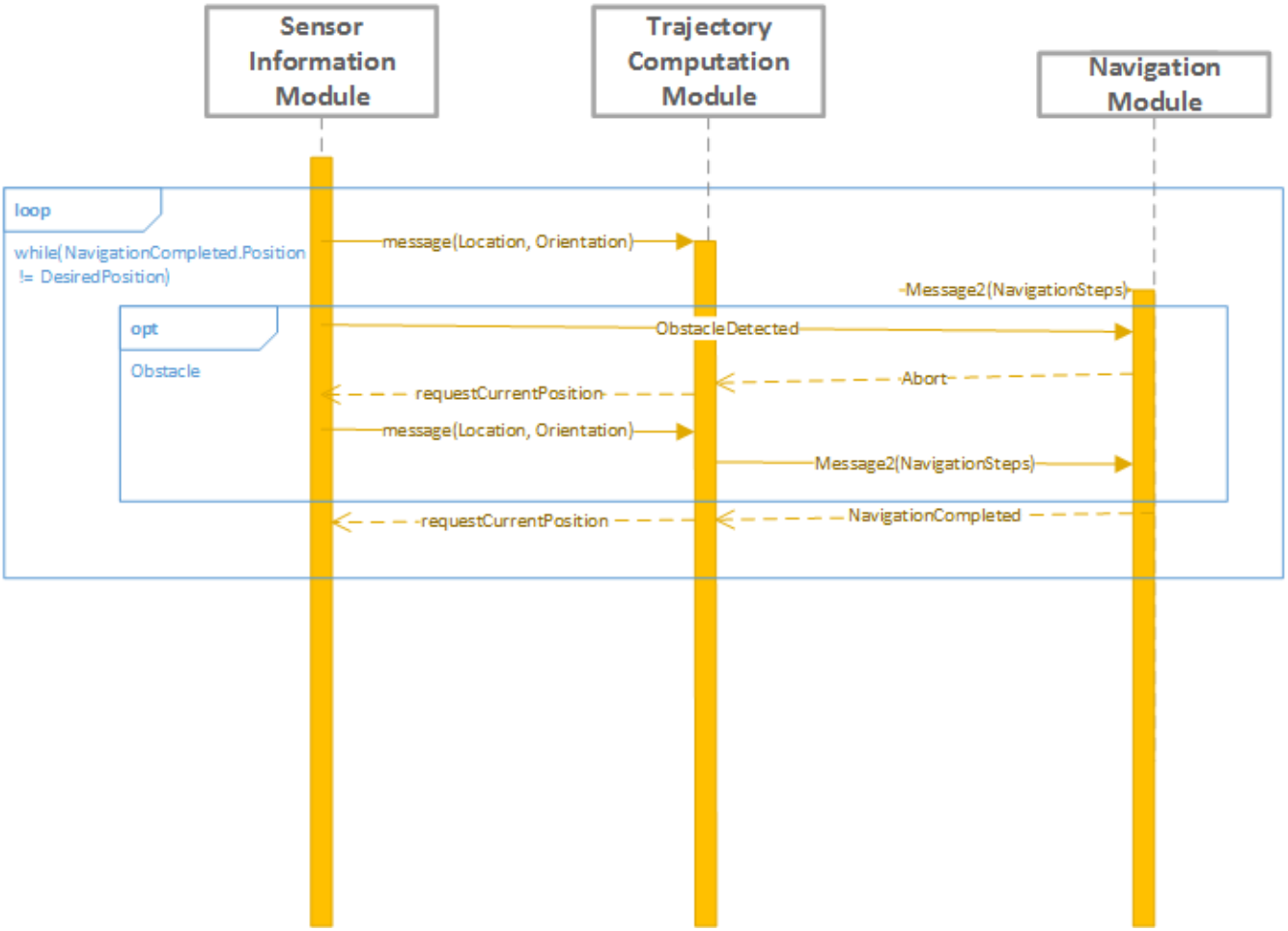
Алгоритъма на модула за навигация може да бъде представен по следната фигура:



Фигура 22

Тъй като алгоритъма се изпълнява постоянно (циклично се извиква функцията му), то всеки път при викане на алгоритъма (и ако колата трябва да продължи да се движи) се проверява дали има засечено препятствие. За да има информация за препятствия, модула за навигация по изчислената траектория получава нотификации от модула за събрана информация от сензорите. Това действие може

да бъде видяно на Фигура 23, съобщението ObstacleDetected, което не е задължително да се случи.



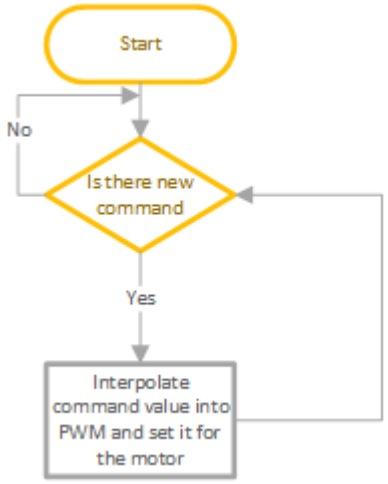
Фигура 23

Навигацията по целевата траектория приключва след като текущата позиция е равна на желаната позиция, като може да има определена граница за допустима грешка.

1.3.4 Модул за управление на хардуера за движение (моторите за движение)

Модулът за управление на хардуера за движение взема като входни данни чисти команди, които са определени от логическите модули за изчисление. Така модула, който управлява хардуера е абстрахиран от всякаква логика на по-високо ниво. Като входни данни модула за управление на моторите получава команди като завой и скорост, които той сам по себе си превръща в PWM за да може да управлява безчетковите мотори. Логиката е проста и се изпълнява циклично, както е оказано

в следващата блок диаграма за всеки мотор (в случая на системата са два безчеткови мотора – за движение напред/назад и за завой):



Фигура 24

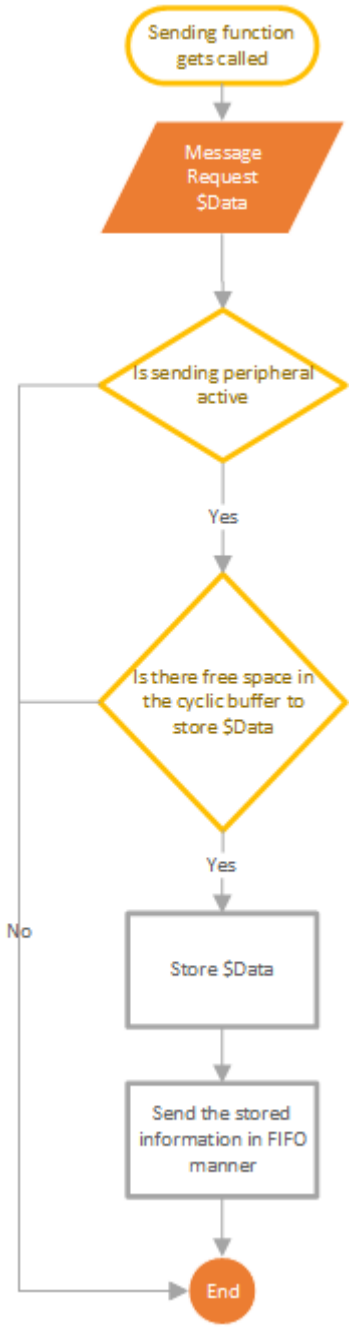
1.3.5 Модул за управление на комуникационния хардуер

Модула за комуникации съдържа функции, които биват извиквани за да изпратят съобщения по желания комуникационен медиум (RS232 или Bluetooth). Тъй като микроконтролера има вградени хардуерни UART комуникационни модули, ако те са конфигурирани правилно е достатъчно софтуера само да напълни комуникационните буфери и UART модулите сами ще предадат желаната информация до съответното целево устройство. Работа на софтуера е да си пази негови комуникационни буфери, които стъпка по стъпка да изпразва към хардуерните UART буфери.

Софтуерните буфери са имплементирани чрез циклични буфери, които ако се напълнят просто спират да приемат информация докато не бъде освободено място за новата такава. Комуникационните буфери се ползват както за комуникиране със системните устройства като компаса и GPS, така и с командващите устройства като Smart Phone с Bluetooth или компютър с RS232. За всеки комуникационен канал е заделен отделен хардуерен модул за комуникация, с цел да не се объркват съобщенията и менажирането им да е по-лесно и удобно. Ако всички видове комуникация се контролираха от един хардуерен UART модул, то между съобщенията предназначени за различни устройства и медиуми за комуникация

трябва да има голямо и вариращо изчакване, тъй като различните комуникационни протоколи поддържат и различни скорости. Това би наложило конфигурирането на ново на UART модула всеки път, когато протокола за комуникация трябва да бъде сменен.

Цикличните буфери са по-подробно описани в глава 1.1. Следната блок диаграма показва в опростен вид как се изпраща съобщение чрез извикване на комуникационна функция.



Фигура 25

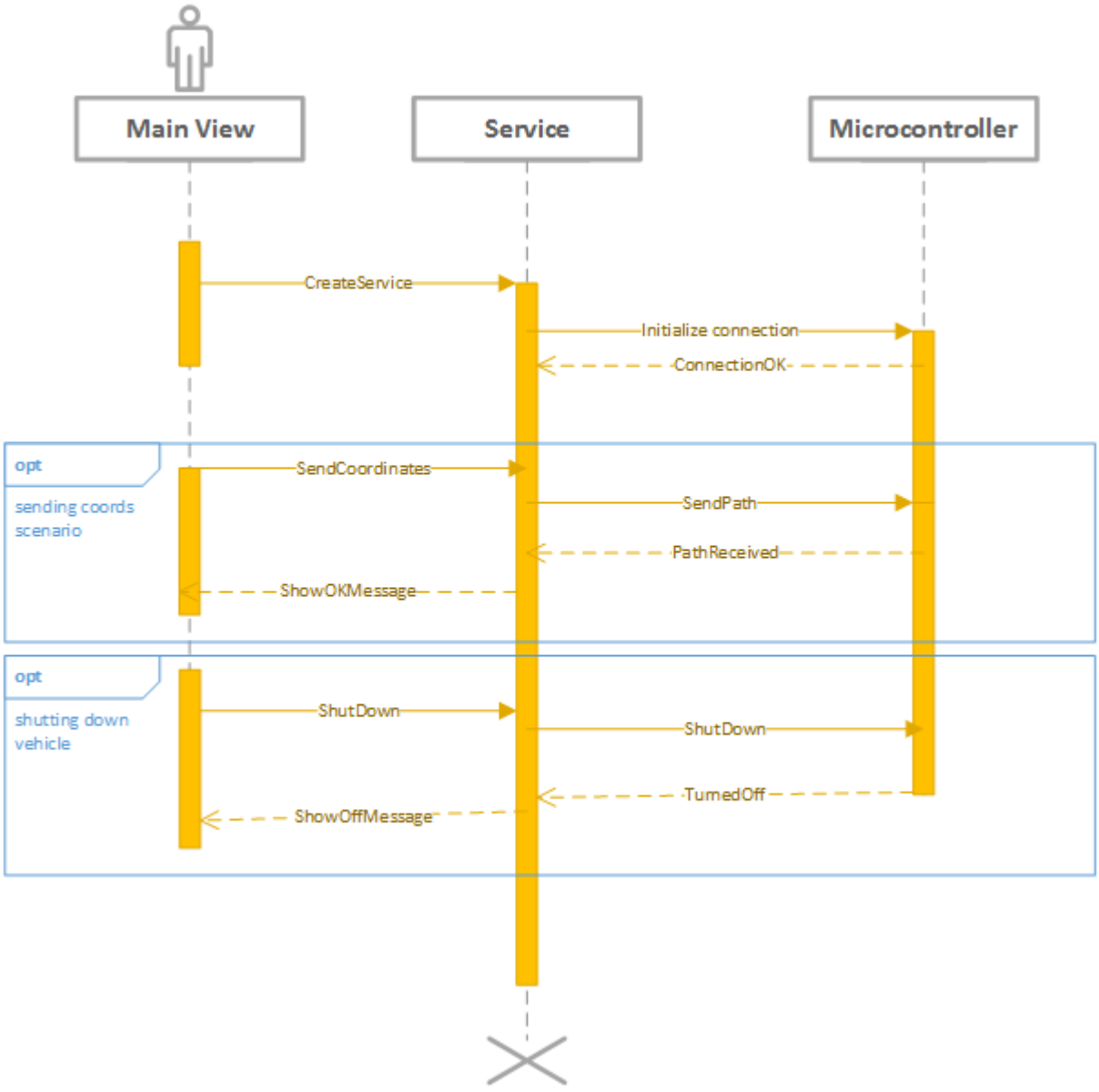
2 Android приложение

Android приложението има за цел комуникация и обмен на данни по безжичен път с микроконтролера. Връзката се осъществява посредством технологията Bluetooth 4.0 и ползва Low Energy стандарта. Протокола, който бива ползван е сериен протокол по Bluetooth, като комуникацията може да се осъществява в двете посоки – от Android устройството към микроконтролера и обратно. Главните цели на приложението за Android са следните:

- Конфигуриране на системата – Задаване на координати – Чрез API предоставено от Google Maps, потребителя може да види текущото местоположение на автомобила (ако GPS приемника е имал възможността да осъществи връзка със достатъчно на брой сателити и да измисли координатите) и да зададе желана дестинация директно върху изобразена карта на Android устройството си.
- Ръчно управление – Преминаване от автономен режим към ръчен режим – по този начин към автомобила биват изпращани команди за управление чрез Android устройството – може да се контролира завиването, движението и скоростта напред/назад, както и светлините на автомобила.
- Аварийно прекратяване на операциите – Ако нещо се обърка и автономно управлявания автомобил изгуби контрол има възможност той да бъде напълно изключен чрез аварийна команда за спиране. След това той трябва да бъде ръчно рестартиран (инициализиране на целия софтуер в начално състояние) за да бъде работоспособен.
- Записване на статистически данни – Android приложението има възможността да изисква статистически данни от автомобила, записвайки средна скорост, измината траектория, засечени грешки по време на работа на автономния автомобил.

2.1 Архитектура на Android приложението.

Android приложението трябва да осъществява постоянна връзка с Bluetooth модула на микроконтролера, когато такава е инициирана. За целта приложението ползва View-та, които изпращат събития (events) към Service, който всъщност осъществява и държи връзката по Bluetooth активна, дори и в момента приложението да се изпълнява на фонов режим или да е пуснато на пауза. Първото Main view има задължението да създаде Service-а, поддържащ комуникацията и след това дори то да бъде затворено, Service-а остава да работи. Следващата фигура илюстрира последователността на събитията за създаване на Service и обмена на данни между микроконтролера, Service-а и View-то:



Фигура 26

2.1.1 Какво представлява Service в Android.

Service е компонент от апликацията, който може да извършва операции, които отнемат много време на фонов режим и не предоставя потребителски интерфейс. Всеки service бива стартиран от друг компонент в апликацията и след като е стартиран той продължава фонова работа, дори и потребителя да превключи активната апликация (тя вече да не е тази, която е активирала дадения service). Като допълнение към създаването, всеки компонент може и да се свърже (bind) към service в неговата апликация. Например service може да бъде използван за извършване на мрежови операции, да възпроизвежда аудио, файлови манипулации или както в случая осъществява постоянна връзка чрез Bluetooth. [37]

Service в Android може да бъде от два типа:

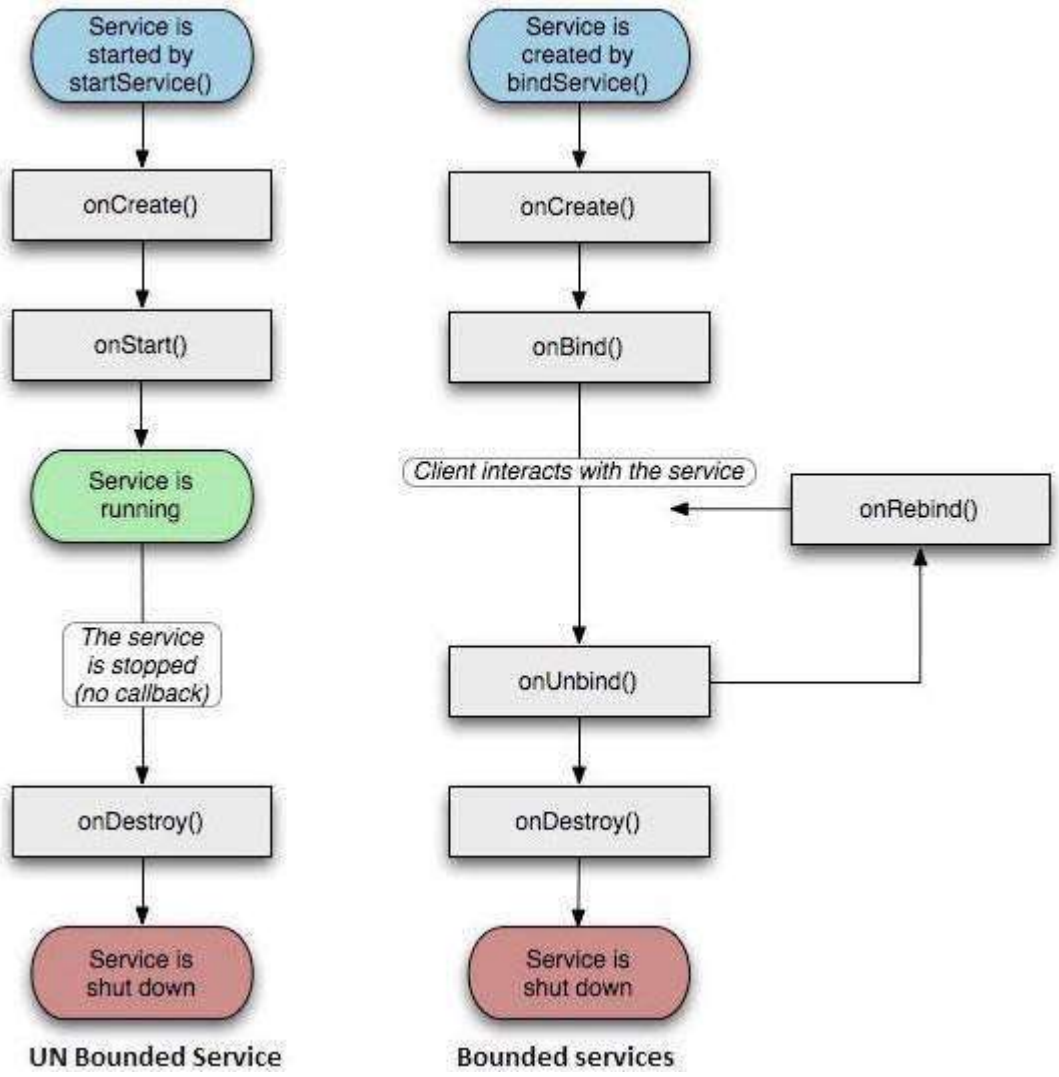
Стартиран (Started) – Service се счита като стартиран, когато компонент от апликацията (като например activity) го стартира чрез функцията startService(). След като бъде стартиран даден service, той може да се изпълнява на фонов режим без ограничение във времето, дори и ако компонента, който го е стартирал вече е унищожен. Обикновено стартиран service изпълнява дадена операция, като не връща резултат на извикалия го компонент. Стартиран service трябва да контролира собствения си цикъл на живот и да се спре сам, когато е наложително – например след като завърши качването на файл. [37]

Свързан (Bound) – Service се счита за свързан, когато компонент от апликацията се свърже към него извиквайки функцията bindService(). Свързан service предлага интерфейс от тип клиент-сървър, който позволява на компонентите да взаимодействат със дадения service – изпращат заявки и получават резултати. Свързания service се изпълнява само докато има свързан компонент от апликацията към него. Няколко компонента може да са свързани към даден service по едно и също време, но когато всичките се разкачат service-а бива унищожен. [37]

Има и вариант service-ите да са едновременно и стартирани и свързани, като за целта първо трябва да бъде извикана startService() функцията и след това всеки

компонент да се свързва чрез bindService() функцията. Именно такъв метод използва Android приложението, тъй като той позволява запазване на стартирания service и съответно установената Bluetooth връзка, дори и апликацията да не е активна, както и удобен интерфейс за комуникация между service-а и компонентите, които го ползват. Единствения вариант в случая, когато service-а може да бъде прекратен е или да получи съобщение от ползващ го компонент за прекратяване (потребителя е избрал бутон за излизане от приложението изцяло) или Android операционната система няма достатъчно свободна RAM памет и унищожава процеси, които са по-маловажни (в момента има процеси, които са по-наложителни и ресурсите се приоритезират за тях). В такъв случай цялата апликация спира работа и връзката с Bluetooth се прекъсва, като се извиква onDestroy() метода от жизнения цикъл на service-а (това може да бъде видяно на Фигура 27). При принудително унищожаване на service-а, той изпраща команда към автономно управлявания автомобил, че Android приложението е спряно и според зависи от режима на работа, автомобила може да прекъсне изпълнение на задачите си или да продължи в автономен режим, докато не достигне целта си (което не е препоръчително, понеже функционалността за аварийно спиране няма да работи, тъй като тя се активира от Android приложението).

Всеки service в Android има собствен цикъл на живот. Той може да бъде обобщен от следната диаграма (тя илюстрира жизнените цикли на стартираните и свързаните services):



Фигура 27

[38]

В Android приложението, тъй като service-а е и стартиран и свързан, първо се изпълняват събитията от стартирането на service-а и после тези при свързването (чак когато бива свързан с даден компонент от приложението). При стартиране, чрез командата onCreate() се инициализират променливите, ползвани от дадения service, след което той преминава в състояние running. В това състояние той изпълнява стейт машина, която управлява Bluetooth модула на телефона. Тази стейт машина получава събития от свързаните към service-а компоненти, които променят нейните състояние – например получава събития за сканиране за bluetooth устройства, за свързване с дадено Bluetooth устройство, за предаване на данни към дадено Bluetooth устройство и т.н.

За да се пращат събития от и към service в текущото Android приложение е нужно дадения service да бъде свързан с компонент. Това бива постигнато чрез bindService() функцията, след викането на която се изпълнява onBind() или onRebind() функцията му. Към тях биват изпращани и референции към повикалите ги компоненти, като по този начин service-а записва референцията към активния компонент, който го ползва и така може да праща събития към него (а не само да приема такива). При пускане на друго приложение или излизане без бутона за изход от приложението, то компонентите, които са свързани към service-а биват пускани на пауза или унищожавани – тези действия извикват onUnbind() функцията на service-а, където той изтрива референцията към компонента, за да не се получи null Reference (използване на несъществуващ обект).

III. ГЛАВА ТРЕТА: ТЕСТОВЕ И РЕЗУЛТАТИ

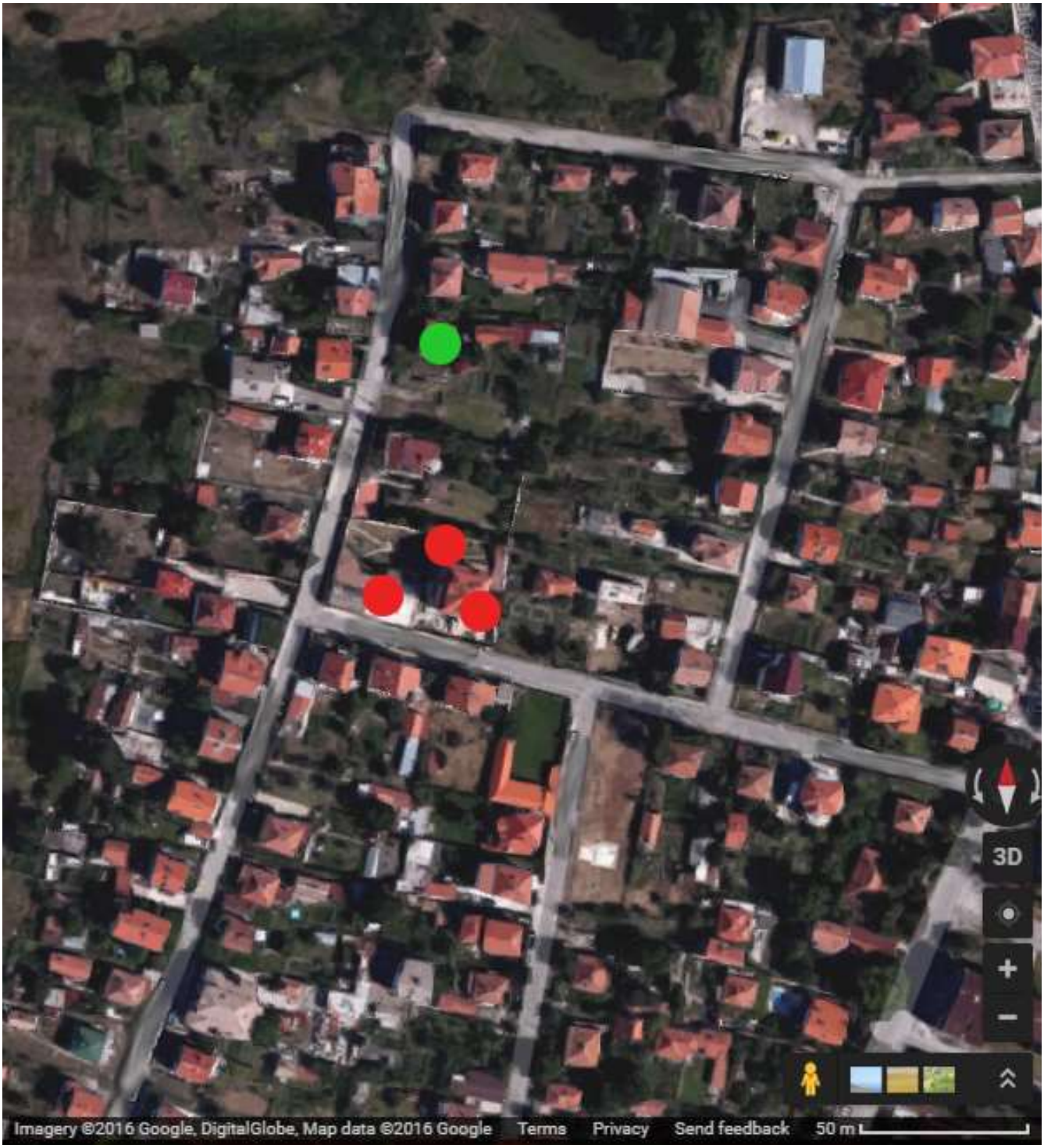
Тестовете са направени на някои хардуерни модули от примерната система, а не на цялата такава. Целта на тестовете е да бъдат получени данни, от които могат да бъдат извлечени изводи за това до колко надеждни и ефективно са избраните модули.

1 Измерване на местоположението

Първата система, която беше тествана е GPS системата. Изпитанията за намиране на GPS координати се провеждат по няколко сценария, като данните се записват на SD карта или директно четени чрез дебъгер от паметта на устройството и после биват обработени на компютър – координатите биват въвеждани в Google Maps и се прави сравнение на истинското местоположение и измереното такова.

1.1 Микроконтролера е вътре в сграда, разположен в близост до прозорец. Населеното място е предимно от къщи и не много високи сгради. В този случай парсираните GPS координати се четат директно от паметта на микроконтролера с debugger веднага след като са обработени. Направени са три измервания на координатите чрез GPS модула в рамките на един ден. Времето по време на

провеждане на теста е ясно с леко заоблачаване. Следната визуализация показва резултатите от теста (за визуализиране е ползвана картографическата услуга Google Maps):

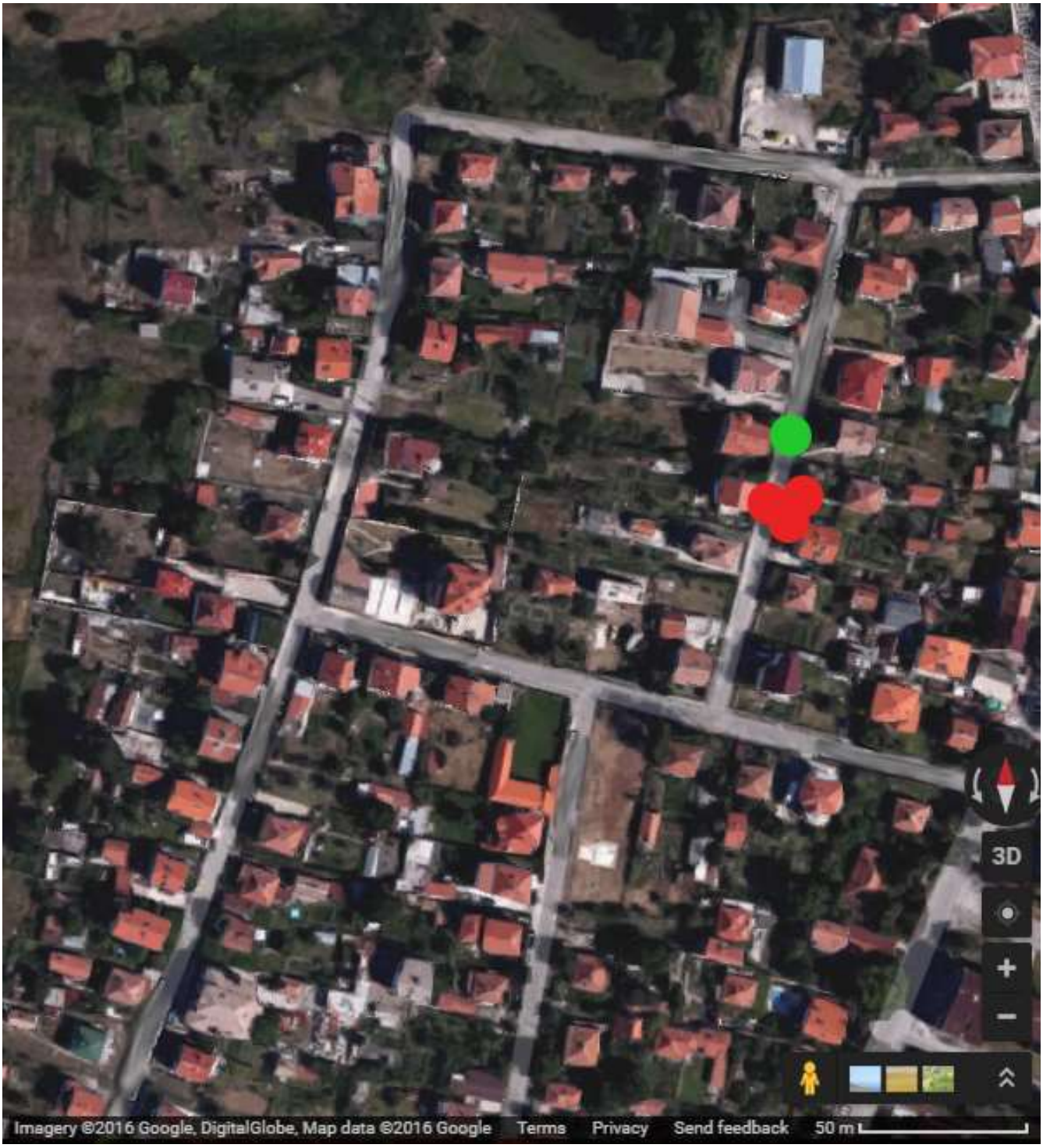


Фигура 28

От измерванията по време на теста може да бъде видяно:

Грешката, която се получава в следствие на позиционирането на GPS модула в сграда е от порядъка на 50 метра, като предимно варира в една и съща посока – в случая Юг. Това е така, понеже GPS Модула вижда по-малък брой сателити, от колкото би видял, ако е извън сграда на открито. За грешката подпомагат и смущенията причинени от отскачане на GPS сигнала в други съседни сгради и преминаването му през медиум различен от въздух (стъкло).

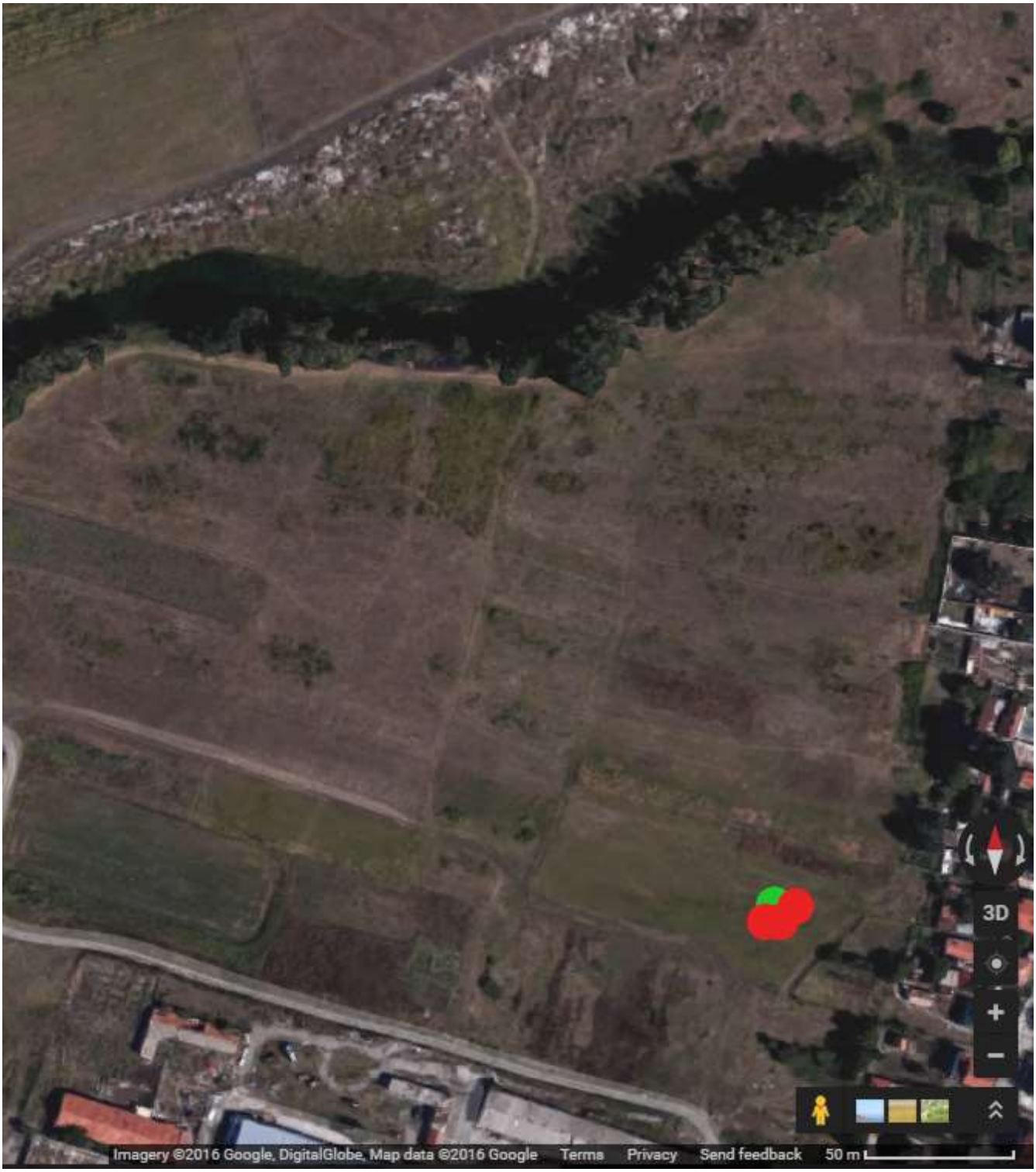
1.2 Устройството се намира на открито в населено място. Населеното място е предимно от къщи и ниско сгради. Измерените GPS координати биват записвани на SD карта, която после бива прочетена с компютър. Всеки един от тестовете в този сценарии също е проведен в рамките на един ден. Времето е слънчево без облаци. Получените резултати могат да бъдат видени на следната визуализация (ползваща картографическата услуга Google Maps):



Фигура 29

На визуализацията може да бъде видяно, че грешката е в рамките на 15-20 метра. Въпреки, че е извън сграда грешка при измерванията се получава поради отскачането на GPS сигнала от близките сгради, което удължава времето му за пътуване и съответно дава лека грешка в измерената позиция.

1.3 Устройството се намира на открито поле, без сгради около него, които да смущават GPS сигнала. Измерените координати се записват на SD карта, като после биват прочитани на компютър и визуализирани чрез Google Maps. Измерванията са проведени в рамките на един ден. Времето е ясно, без облаци. Резултатите могат да бъдат видени на следващата визуализация:



Фигура 30

Както се вижда грешката е в рамките на 7-10 метра максимум. Това са най-точните получени резултати от всички тестове на GPS приемника, тъй като наоколо няма какво да смущава сигнала и се виждат максимален брой GPS сателити при невъзпрепятстващо сигнала време.

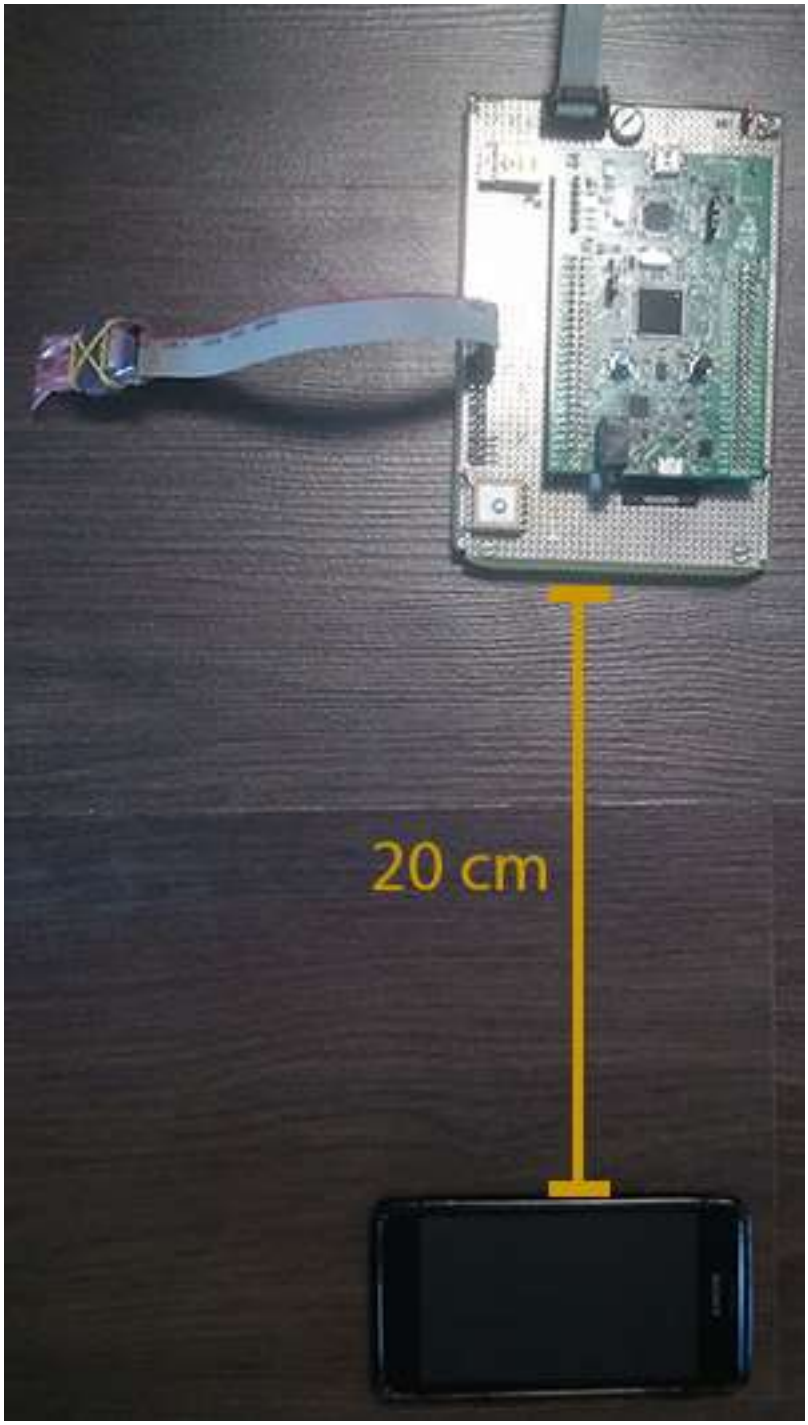
Като обобщение от теста може да се каже, че колкото по-малко препятствия и постройки има около GPS приемника, толкова по-точни резултати дава той.

2 Надеждна комуникация

Надеждността на комуникацията е важна цел във всяка една система, за да може да се гарантира целостта и правилното обработване на данните. При наличието на сигурна и надеждна комуникация може да се разчита на това, че минимален брой от предадените съобщения ще бъдат изгубени и ако това се случи да се гарантира, че системата няма да остане в неработоспособно състояние.

Теста за надеждна комуникация се провежда върху Bluetooth връзката, осъществена между микроконтролера и Smart Phone с операционна система Android, Посредством Bluetooth LE модул прикачен към микроконтролера. Теста се провежда по следния начин:

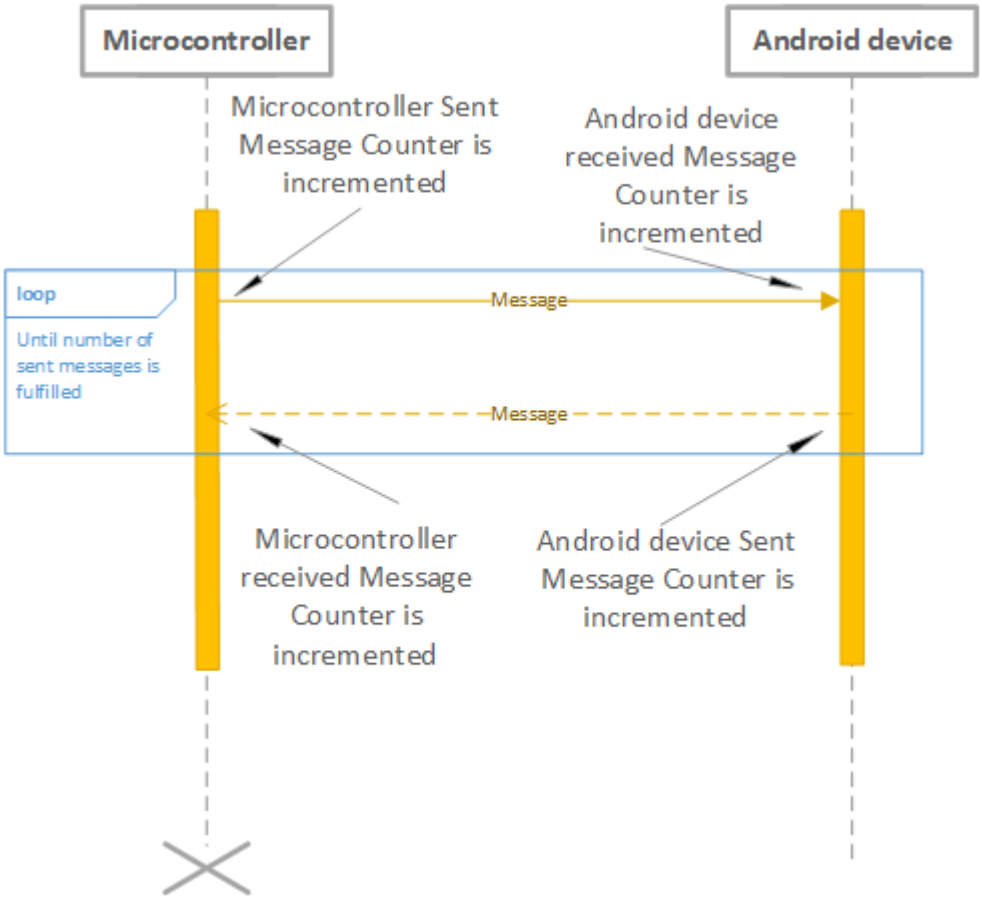
Микроконтролера и Android устройството са на разстояние 15-20 сантиметра един от друг, с включени Bluetooth LE модули, чрез които да предават съобщения помежду си. Между тях няма препятствие и тестовата процедура изглежда както на следната графика:



Фигура 31

Микроконтролера изпраща съобщение и увеличава с единица брояч за изпратените от него съобщения. Android устройството приема съобщението и увеличава с единица брояч за получените от него съобщения. След това Android устройството изпраща съобщение и увеличава с единица брояч за изпратените от него съобщения. След което микроконтролера получава съобщение и увеличава с единица брояч за

получените от него съобщения. Теста се състои общо от четири брояча – по два за всяко от участващите устройства. Този цикъл се повтаря докато устройствата не изпратят предварително конфигуриран брой съобщения. В случая на конкретния тест, броят на съобщенията беше 10 000, като съобщение се пращаше на всяка половин секунда независимо дали такова е било получено от отсрещната страна. Sequence диаграмата на теста е следната:



Фигура 32

Като резултат от теста Bluetooth LE връзката се указа достатъчно надеждна за предаване на указания през малка част от времето, тъй като някои съобщения бяха изгубени. От общо 10 000 изпратени съобщения от Микроконтролера, Android устройството беше получило 9875. Това е над 98% успеваемост. Съобщенията при обратната връзка може да се счита, че са същите на брои (има разлика от 2%), понеже когато връзката е била губена за кратки периоди се случва едното устройство туко-що да е било пратило съобщение и то да е загубено, докато другото

ще прати, когато връзката туко-що е била възстановена. Следователно този вид комуникация е подходящ за изпращане на команди и конфигурационни параметри, но не и за по-дълго траещи комуникационни операции. За да се осъществи наистина надеждна комуникация по Bluetooth ще бъде нужно всяко изпратено съобщение да получава acknowledge от другото устройство – с други думи устройството, до което е адресирано съобщението трябва да потвърди, че го е получило и ако това не се случи съобщението да се изпрати на ново няколко пъти (конфигурируем брой), след което ако все още не се получава съобщение връзката да бъде счетена за нестабилна и да бъде прекратена.

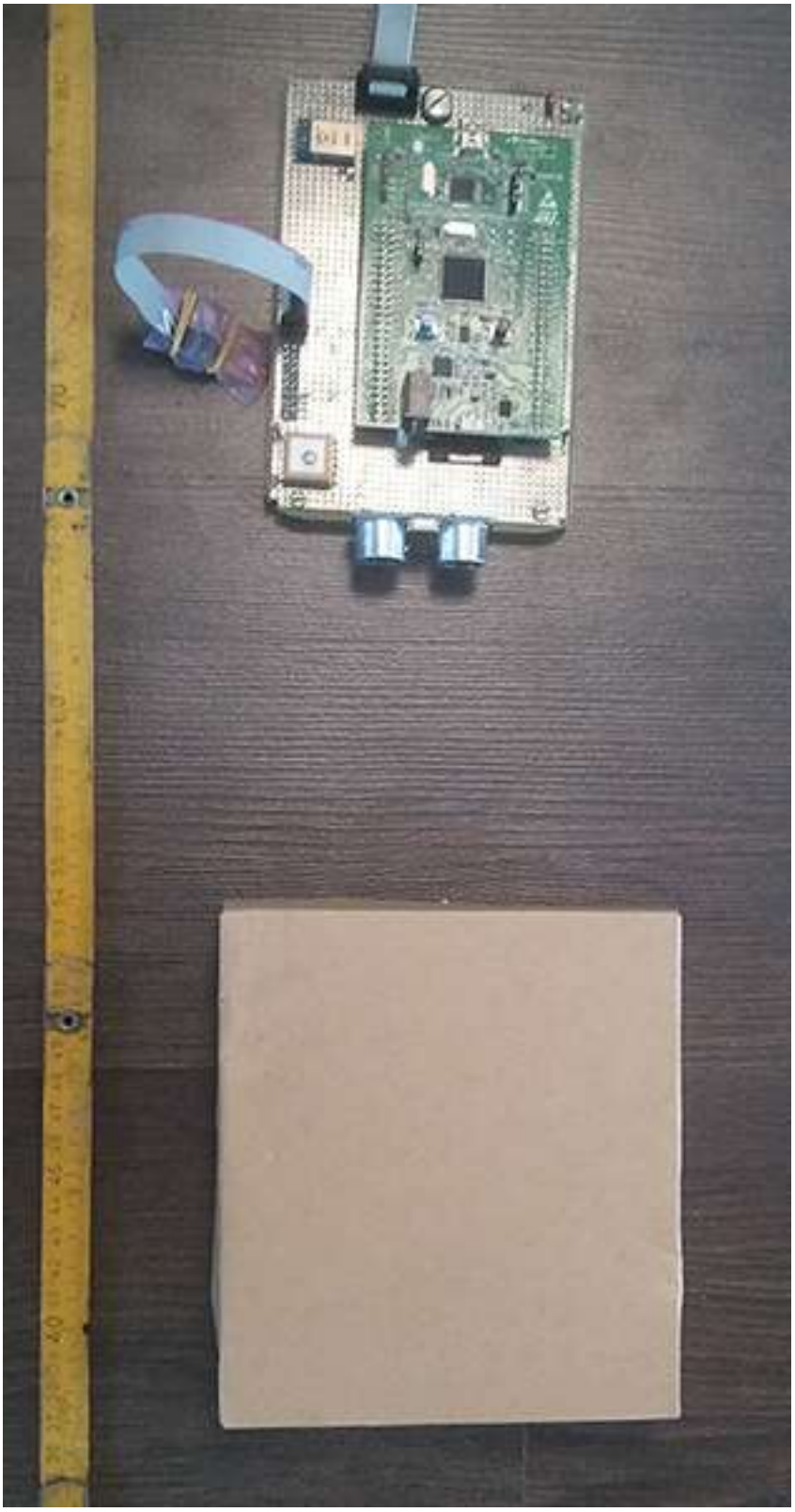
Важно е да се отбележи, че повечето от загубите на съобщения се случиха по време на местене на Android устройството по стаята и скриването му зад стена, която се намира на около 5 метра от микроконтролера с Bluetooth LE модул.

Същия тест беше проведен чрез RS232 комуникация и там няма загуба на съобщения. Това прави RS232 комуникацията подходяща за дълги съобщения, които носят потоци от данни – като например сваляне на нов софтуер върху микроконтролера, постоянен поток от данни за местоположението и др. Тъй като 100% успеваемост и надеждност никога не може да бъде гарантирана в този случай пак е препоръчително да се въведе протокол за комуникация със съобщения за потвърдено получаване на информация (acknowledge messages).

3 Ултразвуков сензор за разстояние

За да бъде тестван ултразвуковия сензор за отчитане на разстоянието бе използвана кутия с размер 25x25 сантиметра, която бе поставена пред сензора. До кутията беше поставен и метър за отмерване на реалното разстояние, като после то бива съпоставено с показанията на ултразвуковия сензор. Кутията се измества в посока далеч от сензора през 5 сантиметра, докато разстоянието не достигне 4 метра (което е максималната измервателна дистанция според спецификациите на ултразвуковия сензор). Към всеки резултат се записва реалното разстояние и измереното такова,

като след това се анализира точността на ултразвуковия сензор. Следващата фигура илюстрира проведеният тест:



Фигура 33

Резултатите от теста показват, че ултразвуковия сензор дава малко повече грешка от колкото е зададено в спецификацията. Стандартната грешка е от +-2-3милиметра, докато при увеличаване на дистанцията над 2 метра грешката в някои случаи скача до сантиметър, като най-голяма грешка е измерена на 3,35 метра при отчитано разстояние от ултразвуковия сензор 336,5 сантиметра (грешка със сантиметър и половина).

4 Заключение от тестовете

Като заключения от тестовете може да се каже, че ще трябва данните от сензорите да бъдат обработени преди да бъдат използвани, понеже не винаги точността им е задоволителна (особено на GPS модула в населени места). Това може да се осъществи чрез осредняване на резултатите в случая на ултразвуковия сензор.

Точността, която трябва да постигнат резултатите от сензорите обаче е само толкова, колкото ще бъде дефинирано като нужно за една такава система – например ако такъв вид автономна машина е предназначено да се движи само по открити полета (за подпомагане на фермерска работа) няма да бъде нужна смяна на GPS модула или някакви други специални обработки на данните от него.

Ако автономния автомобил ще се движи само в определена територия, то до нея може да се сложи втори GPS приемник, на който се знае точното местоположение и по този начин да се измерва грешката в информацията, давана от сателита. След това тази грешка може по безжичен път да бъде изпращана до автономната система. Изваждайки по този начин грешката от GPS-а на автономния автомобил, позицията му може да бъде определена с точност до метър.

Резултатите от комуникацията показват, че безжичната такава е достатъчно стабилна за изпращане/получаване на команди, а кабелната комуникация дори за флашване на нов софтуер. Ако за системата е нужно комуникиране по големи разстояние към нея може да бъде прикрепен и 3G модул, който да ползва интернет както всеки един мобилен телефон, но тогава ще трябва да се обърне внимание и на

по-сериозна защита на комуникацията, понеже атаките ще могат да стават от доста по-отдалечени точки.

ЗАКЛЮЧЕНИЕ

Анализирайки темата беше стигнато до извода, че автономните машини са част от бъдещето. В момента се правят усилия автономните автомобили да станат масови, да се разработят автономни селскостопански машини и военна наземна техника.

Главните предизвикателства пред производителите на такива автономни системи е тяхната безопасност – правилното им реагиране при неопределени и изненадващи ситуации, както и надеждното им навигиране по улиците между автомобили, управлявани от хора, пешеходци, колоездачи и други участници в движението.

Благодарение на все по-модерните сензори, модерните автономни автомобили имат възможността да превърнат заобикалящия ги свят в 3D изображение, по което да се навигират. Добавяйки и компютърно зрение чрез камери информацията, с която разполагат системите е огромна и именно за там идва трудността да се напише софтуер, който да взима точните решения. Някои от имплементациите ползват невронни мрежи, които се учат от грешките си и с течение на времето се подобряват, но за такова решение се изискват огромни средства и сървърна инфраструктура, която да събира резултати от всеки автомобил и да ги обработва подобаващо.

Първото масово навлизане на автономните технологии на потребителския пазар е от компанията за производство на автомобили Tesla в техните Model S. Резултатите от въвеждането на технологията са положителни и автономното управление в автомобилите им продължава да бъде усъвършенствано чрез софтуерни ъпдейти. Гиганти като Google също разработват свои версии на автономните автомобили.

Магистърската теза дефинира минимален набор от модули и софтуер, които са нужни за функционирането на един автономен електрически автомобил. Резултатите от проведените тестове на отделните модули показват, че избраните сензори и комуникационни медиуми са подходящи като минимум и чрез тях може да се извършва автономна операция. На базата на проведените тестове са заложени

и предложения за подобрения в системата, съставена от минимален набор от модули.

Важното заключение е, че при разработката на автономни системи трябва да се наблегне на сигурността както при писането на софтуер, така и при имплементация на защити срещу злонамерени лица.

ИЗПОЛЗВАНИ ИЗТОЧНИЦИ

[1] V. Automobiles, “Volvo vision 2020.” https://www.unece.org/fileadmin/DAM/trans/roadsafe/unda/Sweden_Volvo_Vision_2020.pdf.

[2] J. Greenough, “10 million self-driving cars will be on the road by 2020,” July 2015. http://www.businessinsider.com/report-10-million-self-driving-cars-will-be-on-the-road-by-2020-2015-5-6.

[3] W. F. Inc., “Autonomous car.” https://en.wikipedia.org/wiki/Autonomous_car.

[4] W. F. Inc., “Microcontroller.” https://en.wikipedia.org/wiki/Microcontroller.

[5] W. F. Inc., “Flashing.” https://en.wikipedia.org/wiki/Firmware#Flashing.

[6] W. F. Inc., “Cryptographic hash function.” https://en.wikipedia.org/wiki/Cryptographic_hash_function.

[7] T. Motors, “Your autopilot has arrived.” https://www.teslamotors.com/blog/your-autopilot-has-arrived.

[8] WIRED.com, “Obviously drivers are already abusing teslas autopilot.” http://www.wired.com/2015/10/obviously-drivers-are-already-abusing-teslas-autopilot/.

[9] I. UK, “Tesla autopilot receives latest upgrade: Self-driving feature tweaked in response to real-world use.” http://www.independent.co.uk/life-style/motoring/motoring-news/tesla-autopilot-receives-latest-upgrade-self-driving-feature-tweaked-in-response-to-real-world-use-a6822816.html.

[10] A. Inc., 12 2015. https://static.googleusercontent.com/media/www.google.com/en//selfdrivingcar/files/reports/report-1215.pdf.

[11] A. Inc. https://static.googleusercontent.com/media/www.google.com/en//selfdrivingcar/files/reports/report-annual-15.pdf.

[12] W. Inc., "Google self-driving car." https://en.wikipedia.org/wiki/Google_self-driving_car.

[13] A. Inc., "Google self driving car." https://www.google.com/selfdrivingcar/how/.

[14] W. Foundation, "Driverless tractor." https://en.wikipedia.org/wiki/Driverless_tractor.

[15] H. W. N. H. N. M. R.-J. J. Blackmore, B. S.; Griepentrog, "Development of a deterministic autonomous tractor.," tech. rep., The Royal Veterinary and Agricultural University (KVL), Dept. of Agricultural Sciences Frederiksberg / Copenhagen, Denmark, 2004.

[16] FarmShow. http://www.farmshow.com/view_articles.php?a_id=1458t.

[17] W. F. Inc., "Darpa grand challenge." https://en.wikipedia.org/wiki/DARPA_Grand_Challenge#2004_Grand_Challenge.

[18] D. ALEXANDER, "Usa defense robots," 3 2012. http://www.reuters.com/article/us-usa-defense-robots-idUSBRE84805N20120509.

[19] E. Ackerman, "Lockheed's robotic trucks pass real-world military convoy test," 2 2014. http://spectrum.ieee.org/automaton/robotics/military-robots/lockheeds-robotic-trucks-pass-real-world-military-convoy-test.

[20] 2025AD, "Us army is counting on self-driving trucks," 2 2016. https://www.2025ad.com/in-the-news/news/army-to-test-driverless-trucks/.

[21] W. F. Inc., "Bluetooth," January 2016. http://en.wikipedia.org/wiki/Bluetooth.

[22] "Bluetooth specification version 4.0," June 2010.

[23] W. F. Inc, "Scatternet," Jan. 2016. https://en.wikipedia.org/wiki/Scatternet.

[24] B. SIG, “Bluetooth technology basic.” http://www.bluetooth.com/Pages/Basics.aspx.

[25] DigiKey. http://www.digikey.com/en/articles/techzone/2011/dec/bluetooth-low-energy-technology-makes-new-applications-possible.

[26] J. Decuir. http://chapters.comsoc.org/vancouver/BTLER3.pdf.

[27] NewCircle. https://newcircle.com/s/post/1553/bluetooth_smart_le_android_tutorial.

[28] Adafruit. https://learn.adafruit.com/introduction-to-bluetooth-low-energy/gatt.

[29] W. F. Inc, “Bluetooth low energy.” http://en.wikipedia.org/wiki/Bluetooth_low_energy.

[30] W. F. Inc., “Rs 232.” https://en.wikipedia.org/wiki/RS-232.

[31] Quectel, “Quectel l80 compact gps module, integrated with patch antenna.” http://www.quectel.com/UploadFile/Product/Quectel_L80_GPS_Specification_V1.1.pdf .

[32] Micropik, “Hc-sr04.” http://www.micropik.com/PDF/HCSR04.pdf.

[33] Olimex, “Mod-hmc5883l.” https://www.olimex.com/Products/Modules/Sensors/MOD-HMC5883L/open-source-hardware.

[34] “Misra c 2004.”

[35] G. Baddeley, “Gps - nmea sentence information,” 2001. http://aprs.gids.nl/nmea/.

[36] W. F. Inc., “Schmitt trigger.” https://en.wikipedia.org/wiki/Schmitt_trigger.

[37] A. Inc., “Android services.” http://developer.android.com/guide/components/services.html.

[38] TutorialsPoint, “Android - services.” http://www.tutorialspoint.com/android/android_services.htm.