

УНИВЕРЗИТЕТ У БЕОГРАДУ

МАТЕМАТИЧКИ ФАКУЛТЕТ

МАСТЕР РАД

Развој софтвера за обраду и приказ информација у моторним возилима

Студент:
Тијана ЈОРДАНОВ

Ментор:
Проф. др Владимир
ФИЛИПОВИЋ

8. септембар 2018.



Ментор:

Проф. др Владимир Филиповић

Математички факултет, Универзитет у Београду

Чланови комисије:

Проф. др Филип Марић

Математички факултет, Универзитет у Београду

Проф. Доц. др Александар Картељ

Математички факултет, Универзитет у Београду

Датум одбране:

Садржај

1	Увод	1
2	Архитектура аналогне контролне табле	3
3	Raspberry Pi	7
4	Ардуино	11
5	Спецификација захтева и анализа система	14
5.1	Покретање возила	15
5.2	Убрзавање	16
5.3	Сигнализација	17
5.4	Детекција проблема	17
5.5	Кочење	20
5.6	Заустављање возила	20
6	Дизајн решења	21
7	Имплементација решења	24
7.1	Конфигурација	25
7.2	Упис у дневник	26
7.3	Обрада слика	28
7.4	Контрола GPIO пинова	29
7.5	Комуникација по I ² C протоколу	30
7.6	Исцртавање елемената	34
8	Тестирање	39
8.1	Тестирање графичког корисничког интерфејса	39
8.2	Тестирање пинова опште намене	43
8.3	Тестирање комуникације по I ² C протоколу	43
9	Закључак	45
	Литература	46

1 Увод

Човек је одувек имао потребу за брзим и једноставним транспортом. О значају транспорта говори и чињеница да се проналазак точка сматра једним од најзначајнијих открића за људску цивилизацију. Хиљадама година је за превоз коришћена снага животиња док у 18. веку нису конструисана прва возила која је покретала снага паре, а већ у 19. веку настају прва возила која покреће мотор са унутрашњим сагоревањем. Овај мотор изумео је немац Николаус Ото, а његов сународник, Карл Бенз, 1885. године конструисао је прво возило које покреће мотор са унутрашњим сагоревањем. Возило које је Карл Бенз конструисао сматра се првим моторним возилом налик данашњем. Бензу је патент за овај проналазак одобрен 1886. године, а аутомобил *Velo*, скраћено од *Velociped*, који је произведен крајем 19. века у фабрици *Benz & Co.* може се сматрати првим аутомобилом масовне производње.[9] У периоду од 7 година произведено је 1200 возила *Velo* и *Velociped Comfortable*.[2]

Моторна возила представљала су престиж све док 1908. године Хенри Форд није започео са масовном производњом возила познатог као Модел Т. Ово возило, произведено на покретној траци, било је приступачно, издржљиво и лако за одржавање. Данас су возила део свакодневног живота. Због разноврсне намене класификована су у различите категорије, а у производњи је највећи број путничких возила. Према подацима Интернационалне организације произвођача моторних возила у 2016. години у свету је произведено 94 976 569 моторних возила, при чему је удео путничких возила 72 105 435. Број регистрованих возила је много већи и 2015. године је износио више од 1.2 милијарде са уделом путничких возила већим од 947 милиона.[1]

Основни принцип рада моторних возила је једноставан: паљењем возила покреће се мотор који обезбеђује енергију за покретање точкова. Расхладна течност спречава да се мотор прегреје, а возач управља возилом. Иако је принцип рада остао исти као крајем 19. века, компоненте које се налазе у данашњим возилима су много напредније. Механичке компоненте су годинама усавршаване, а развојем електронике многи механички уређаји су замењени електронским. Разлози увођења електронских компоненти су заузимање мање простора, мања тежина, бољи квалитет, могућност контроле рада мотора са циљем смањења потрошње горива и мањи трошкови производње. Електронске компоненте пружиле су и нове функционалности за повећање безбедности и комфора возача. Спектар нових могућности је широк: од електронског мерења растојања и навигације до масажних седишта и система за ароматерапију.

Софтвер који је намењен системима у моторном возилу мора да буде поуздан, прецизан, брз. У овом раду је описан процес развоја прототипа софтвера за обраду и приказ информација у моторном возилу. Прототип представља ди-

гиталну верзију контролне табле и зато је на почетку рада описана аналогна контролна табла: њени елементи и њихов начин рада. Развијени прототип извршава се на *Raspberry Pi* уређају, па су у једном поглављу описане могућности овог рачунара. Поглавље 5 садржи захтеве које софтвер мора да испуњава и бави се анализом система, а највећи део рада описује дизајн и имплементацију развијеног прототипа. На крају рада описани су изведени тестови.

2 Архитектура аналогне контролне табле

Моторно возило је сложен систем изграђен од низа механичких, електричних и електронских компоненти. Све ове компоненте повезане су тако да чине стабилан систем. Како би та стабилност била очувана током вожње, неопходно је да возач прати стања уређаја и прилагоди им своје активности. Информације о уређајима су доступне возачу на контролној табли. Контролна табла се налази испред возача, а сигнали на њој су дизајнирани тако да не ометају возача док управља возилом, али да буду довољно упадљиви у случају детектовања неког проблема. Исправним тумачењем ових сигнала и реаговањем на њих возач може да спречи настанак озбиљних кварова или да избегне несрећу.

Контролна табла је главни извор информација о тренутном стању возила. На њој су смештени контролни инструменти и контролно - сигналне лампице. Контролни инструменти су најчешће округли са скалом дуж обода и казаљком. Они пружају возачу стални увид у вредност контролисане величине.

Лампице на контролној табли служе за обавештења и упозорења и према томе се деле у две групе: лампице које указују на исправност уређаја и лампице које обавештавају да је одређени уређај активан [6]. Свака лампица има сијалицу чији интензитет је подешен тако да не омета возача. Боја сијалице има посебно значење: зелена означава обавештење, наранџаста упозорење и црвена указује на озбиљан проблем. Лампица се пали када се догађај на који се она односи јави, а неке лампице које се односе на озбиљне проблеме трепере када су активне како би што пре привукле пажњу возача. Контролна лампица која означава исту појаву не мора исто изгледати код свих произвођача. Разлике постоје у зависности од произвођача, марке возила или дела света за који је возило произведено као што се види у примеру на слици 1. Важно је да возач зна да исправно тумачи ознаке са контролне табле због безбедности током вожње и како не би дошло до квара возила услед неадекватног руковања.



(I) САД



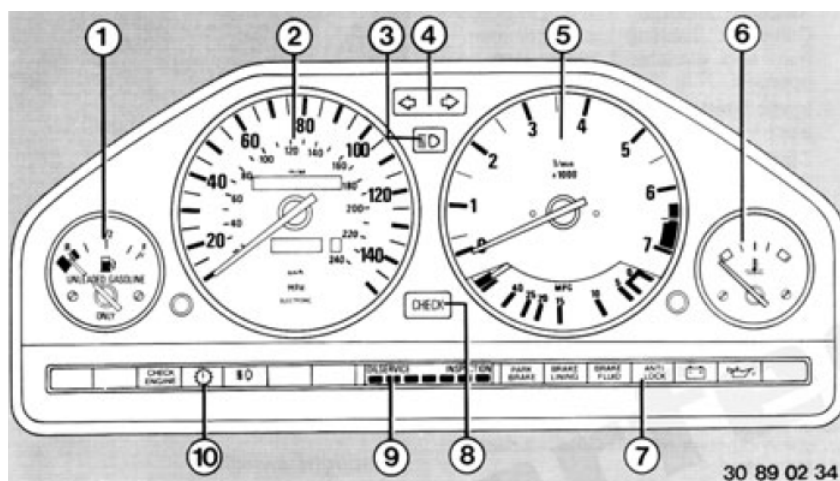
(II) Канада



(III) Европа

Слика 1: Пример контролне лампице за престанак рада АБС-а (енг. *Anti-lock braking system*). Свака лампица намењена је одређеном тржишту.

Тип возила, марка и модел не условљавају само изглед елемената на контролној табли већ и њихов састав. Пример контролне табле дат је на слици 2 на којој су елементи табле означени бројевима. Сваки број одговара редном броју под којим је елемент описан у наставку текста.



Слика 2: Контролна табла

1. Показивач нивоа горива - приказује ниво горива у резервоару. На скали показивача су истакнуте ознаке за пун и празан резервоар и казаљка. Казаљка је повезана на биметалну траку која је повезана са отпорником. При повећању отпора мање струје пролази кроз намотај и биметална трака се хлади. Услед хлађења трака се скупља и помера казаљку ка ознаци за празан резервоар. На мерачу је и лампица која сија жуто када у резервоару остане 8 литара горива или мање [7].
2. Брзиномер - ово је показивач у облику сата са казаљком која приказује брзину и два одометра. Брзиномер са слике приказује тренутну брзину до максималне. На слици је та максимална брзина 140 *milja/h* односно око 220 *km/h*. Изнад и испод казаљке брзиномера са слике налазе се одометри. Одометри приказују број пређених километара. Већи одометар приказује колико је километара возило прешло од првог покретања и његова вредност се не може мењати. Мањи одометар приказује пређени пут, а по потреби се може ресетовати. Дугме за постављање вредности мањег одометра на нулу на слици је приказано између брзиномера и показивача нивоа горива.

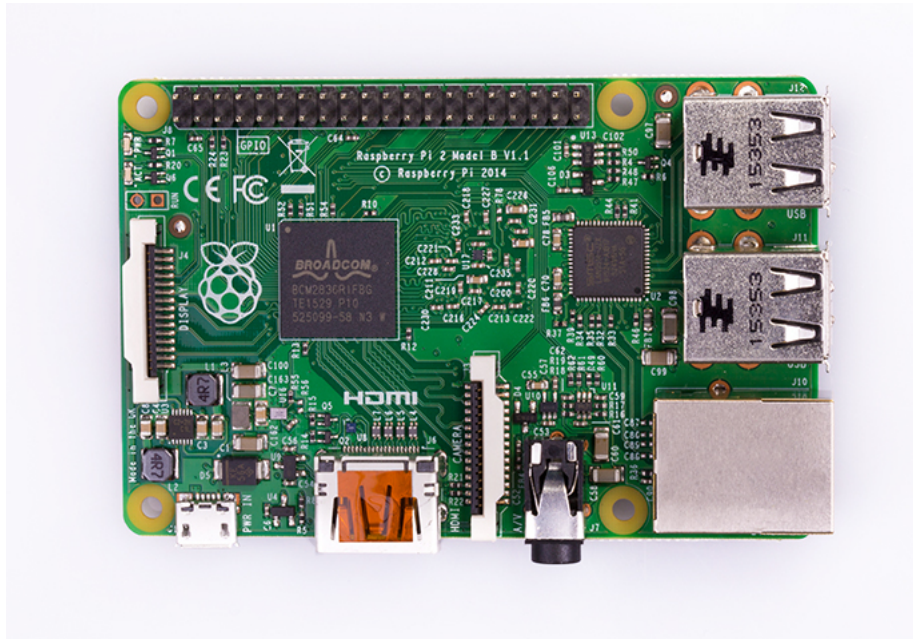
3. Контролна лампица за дуго светло - ова лампица светли ако су упаљена дуга светла и према прописима мора бити присутна на контролној табли [6]. Лампица користи исто напајање као дуго светло па када се ручицом поред волана упали дуго светло, пали се и лампица.
4. Показивачи правца - обавештавају возача да је упаљен сигнал за промену правца. Пале се ручицом поред волана, заједно са мигавцима. Лампица добија напајање од аутомата за мигавце па и она трепери као и мигавци.
5. Бројач обртаја - назива се још и обртомер или тахометар. Он приказује број обртаја које направи мотор у минути. Ово мерило је у облику сата са казаљком и скалом чији је почетак у нули, а максимална вредност варира зависно од возила. У примеру са слике максимална вредност је 7. Ове вредности представљају хиљаде обртаја у минути. Део скале је означен црвеном бојом. Ако се казаљка бројача налази у овој зони, то треба да упозори возача да промени степен преноса. Када је казаљка у црвеној зони аутоматски се смањује убризгавање горива како би се ограничила максимална брзина.
6. Показивач температуре расхладне течности - улога расхладне течности је да хлади мотор током његовог рада. Уколико се ова течност прегреје, она губи своју функцију и ако се кретање возила настави постоји опасност да ће доћи до великих оштећења мотора. Као расхладна течност користи се вода. Показивач температуре расхладне течности има скалу са две обојене зоне и подеоцима између њих. Плава зона означава да мотор још није достигао своју нормалну температуру. Нормална температура се приказује на средини скале. Црвена зона показује да је мотор прегрејан, а температура расхладне течности је прешла дозвољену границу. Показивач температуре расхладне течности приказује температуру коју мери термостат.
7. Индикатори упозорења - ови индикатори односе се на различите зоне возила. У свакој зони постављени су сензори. Уколико сензори детектују да нека компонента не ради по прописаној спецификацији, лампица се пали. Пошто ове индикаторе активирају сензори, приликом сваког стартовања мотора сви индикатори се пале на кратко како би се проверило да ли су исправни. Ако се неки индикатор не упали, потребно је заменити сијалицу, а ако остане упаљен то значи да се јавио проблем у области на коју се односи. На слици у ову групу индикатора сврстани су:
 - Контролна лампица за положај ручне кочнице - лампица се пали да обавести возача да је ручна кочница подигнута. Укључује се микропрекидачем који се налази испод ручне кочнице.

- Индикатор истрошености кочионих плочица - уколико су кочионе плочице превише похабане лампица светли црвено.
 - Индикатор нивоа кочионе течности - ако лампица сија црвено током вожње, то значи да је ниво кочионе течности низак и возило треба што пре зауставити.
 - АБС индикатор - паљење лампице током вожње означава да је АБС престао са радом, али су кочице и даље у функцији.
 - Индикатор стања акумулатора - ако се лампица упали током вожње, означава проблем са допуњавањем акумулатора: батерија је скоро празна или алтернатор не производи довољно електричне енергије да пуни батерију и снабдева ауто током вожње.
 - Индикатор за притисак моторног уља - подмазивање мотора неопходан је услов за његово нормално функционисање. Циркулацију уља обезбеђује пумпа чији рад се контролише притиском уља. Ако нема довољно уља у мотору притисак је нижи, контакти прекидача су затворени и лампица светли црвено. Наставак вожње након паљења ове лампице може узроковати озбиљне проблеме са мотором.
8. Дугме за проверу (eng. *Check*) - пали све индикаторе и иницира проверу светла на регистарским таблицама, стоп светла, обореног и задњег светла и нивоа расхладне течности, течности за прање и моторног уља. Након провере остају упаљени само они индикатори који указују на детектоване проблеме.
9. Сервисни индикатор - ова лампица обавештава да је потребан редован сервис и тада сија зелено или жуто. Уколико сија црвено термин за сервис је премашен. Лампица се ресетује приликом сервисирања возила.
10. Индикатори упозорења:
- Индикатор за проверу мотора - пали се када се детектује проблем у раду мотора.
 - Индикатор који се односи на аутоматски мењач.
 - Контролна лампица за светла за маглу - показује да ли су светла за маглу упаљена.

3 Raspberry Pi

Raspberry Pi је рачунар малих димензија и приступачне цене који је развијен у образовне сврхе. Направила га је група професора са Универзитета у Кембриџу која је потом основала Raspberry Pi фондацију која се бави промовисањем програмирања и едукацијом. Идеја је била да се направи приступачан алат који би будућим кандидатима за упис омогућио да код куће усаврше своја практична знања из програмирања. На разговору би испитивачи могли да питају кандидате које су све пројекте развили помоћу овог рачунара, а одговори би им помогли у процесу селекције [8]. Како би цена рачунара била што нижа, циљ је био искористити што више уређаја које домаћинства већ поседују. Тако овај рачунар не захтева посебан монитор већ поседује HDMI порт и порт за аналогни дисплеј којим се може повезати са телевизором са катодном цеви. За пуњење уређаја користи се USB кабл који је 2012. године постао стандардни пуњач мобилних телефона па је и он присутан у многим домаћинствима. Raspberry Pi поседује и портове за миша и тастатуру, USB портове и пинове опште намене (eng. *General Purpose Input/Output*, *GPIO*). Због приступачне цене, малих димензија и великог броја GPIO пинова које кућни рачунар не поседује Raspberry Pi је од објављивања био веома тражен. Данас постоји неколико модела овог рачунара који се разликују по брзини процесора, величини RAM меморије, броју и врсти портова. Raspberry Pi на ком ће се извршавати софтвер који је предмет овог рада је Raspberry Pi 2 Model B. Овај модел припада другој генерацији Raspberry Pi рачунара и избачен је на тржиште у фебруару 2015. године [3]. То је први Raspberry Pi са 1 GB RAM меморије. Његове димензије су 85.60mm x 56mm x 21mm. Процесор на овом моделу је ARM Cortex-A7 од 900 MHz, а портови су исти као на претходном моделу и виде се на слици 3.

Главна меморија рачунара Raspberry Pi налази се на SD картици која мора имати барем 2 GB јер је на њој смештен оперативни систем. Препоручено је да се оперативни систем инсталира коришћењем NOOBS (eng. *New Out Of Box Software*) алата који нуди избор оперативног система и једноставан процес инсталације. Оперативни систем прилагођен Raspberry Pi уређају је Raspbian који се заснива на Debian дистрибуцији Linux оперативног система. Након инсталације оперативног система може се програмирати на било ком програмском језику који може да се компајлира на ARMv7. Raspberry Pi фондација препоручује Python, али се могу користити и C, C++, Java, Scratch, Ruby.



Слика 3: Raspberry Pi 2 Model B

Писање програма за рад са GPIO пиновима може се реализовати на више начина. Први је директан приступ GPIO регистрима који носи високи ризик од јављања конфликта уколико више процеса приступа истом пину. Друга опција подразумева коришћење неке од библиотека наменски креираних у те сврхе као што су *WiringPi* и *pigpio*. Трећа опција је приступ преко датотека. За сваки GPIO пин постоји директоријум у коме свака датотека чува вредност једног својства као што су смер и вредност. Ови директоријуми налазе се на путањи `/sys/class/gpio/`. Иницијално у овом директоријуму су само датотеке `export` и `unexport` и директоријум `gpiochip0` задужен за GPIO контролере. Да би директоријум са конфигурационим датотекама за одређени пин био креиран, неопходно је број пина уписати у `export` датотеку. Постоје две различите нумерације пинова. Први тип нумерације је по физичком положају. GPIO пинови су поређани на плочи у два реда и оивичени белом правоугаоном линијом. Један ћошак те линије је заобљен и то је место на ком почиње нумерација. Пин који се налази поред заобљеног угла носи физички број један, њему суседни пин у другом реду има физички број два. Нумерација свих пинова може се видети на слици 4 у правоугаонику уоквиреном црвеном бојом. Поред физичких бројева пиновима су додељени и VCM бројеви. VCM број односи се на број канала на *Broadcom SOC* (eng. *system on chip*) интегрисаном колу и на слици 4 су ови бројеви приказани у колонама са стране поред назива GPIO. Приступ пиновима преко датотека

користи BCM број за идентификацију пина.

На Raspberry Pi уређају постоји 40 пинова. Они се могу поделити на пинове опште намене, уземљење, пинове са сталним напоном од 3.3V и 5V. Пинови опште намене имају два могућа стања: стање са напоном (eng. *high*) и стање без напона (eng. *low*). Стање са напоном подразумева напон од 3.3 V и сваки већи напон може да оштети уређај. Ови пинови погодни су за контролу уређаја детектовањем сигнала.

Неки од пинова опште намене повезани су на магистрале за комуникацију по различитим протоколима и тако омогућују пренос веће количине информација. Магистрале на које су повезани пинови су: UART (eng. *Universal Asynchronous Receiver/Transmitter*), I²C (eng. *Inter-Integrated Circuit*), SPI (eng. *Serial Peripheral Interface*).

UART протокол врши серијски пренос података. За имплементацију овог вида комуникације потребна су два комуникациона канала. Размена података је асинхрона што значи да се не користи часовник. Контрола података врши се помоћу почетног, зауставног и бита парности који се додају сваком пакету. Да би се размена података успешно реализовала неопходно је да фреквенција преноса буде приближно једнака на оба уређаја. Фреквенција преноса представља број битова послатих у секунди и зове се бодовна брзина (eng. *baud rate*). Разлика у бодовној брзини између два уређаја не сме бити већа од 10%. У једној итерацији потребно је послати 10 битова: почетни бит, зауставни бит и 8 битова података. За слање сваког од њих потребно је 10 % укупног времена. Ако претпоставимо да је разлика у бодовној брзини између уређаја 10 % тада ће се један бит јавити раније или ће бити пропуштен. У овом протоколу разликују се главни уређај (eng. *master*) и споредни уређај (eng. *slave*). Главни уређај управља комуникацијом и одређује да ли ће да шаље или прима податке. Споредни уређај одговара на захтеве главног уређаја. Недостаци овог протокола су максимална величина од 9 битова података по пакету и то што не подржава више главних или споредних уређаја у једном систему.

SPI је протокол за синхрону серијску размену података. За имплементацију овог вида комуникације потребна су 4 комуникациона канала: MOSI (енг. *Master Out / Slave In*), MISO (енг. *Master In / Slave Out*), SCK (енг. *Serial Clock*) и SS (енг. *Slave Select*). Прва два комуникациона канала користе се за размену података, трећи за слање сигнала часовника, а четврти за одабир споредног уређаја који ће учествовати у комуникацији. У комуникацији може да учествује само један главни уређај. Он је задужен за генерисање сигнала часовника. Споредних уређаја може бити више. Оваква комуникација је бржа од асинхроне, али захтева више комуникационих канала и за сваки споредни уређај потребан је посебан SS комуникациони канал.

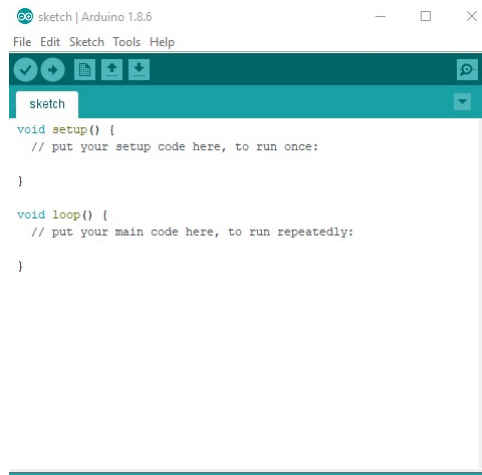
I²C је серијски пртокол који користи 2 комуникациона канала. Један ка-

4 Ардуино

Ардуино је хардверско софтверска развојна платформа отвореног кода. Развијен је од стране Ivera Interaction Design института и баш као и Raspberry Pi био је намењен студентима. Студенти Ivera института су уметници и дизајнери па је зато приликом развоја платформе посебна пажња посвећена томе да она од корисника не захтева висок ниво знања из области електронике и програмирања. Управо због једноставног коришћења ардуино је стекао велику популарност. Како би се што више прилагодио потребама корисника, развијени су различити типови ардуино платформе. Неки од најпопуларнијих чланова ардуино породице су Ардуино UNO - најкоришћенија платформа која се често препоручује почетницима, LilyPad ардуино који је намењен ношењу на одећи, Ардуино Mega платформа са великим бројем пинова, итд.

Главни хардверски део ардуина је микроконтролер. Док већина микроконтролера захтева посебан хардвер како би се нови код спустио на платформу, ардуино омогућује повезивање са рачунаром преко USB кабла. За рачунар је доступно Ардуино развојно окружење (енг. *Arduino IDE, Integrated Development Environment*) које је намењено развоју кода и спуштању софтвера на уређај.

Језик на ком се код развија је поједностављена верзија C++ програмског језика и назива се *Wiring*.^[5] Сваки програм има две главне функције које су приказане на слици 5. То су функције `setup()` и `loop()`. Функција `setup()` извршава се приликом паљења уређаја. У њој се врши иницијализација података. Извршавање функције `loop()` се непрестано понавља све док је уређај упаљен. У овој функцији се налази главна логика програма.



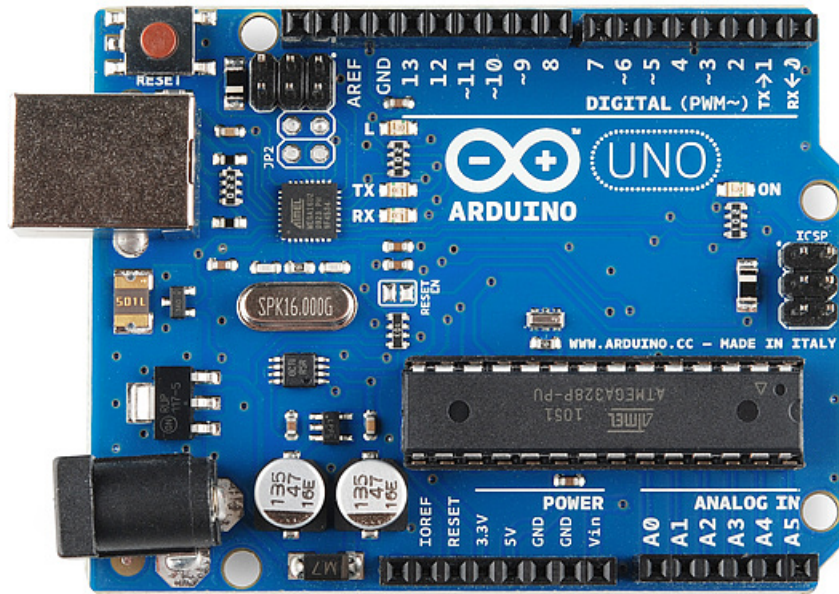
```
sketch | Arduino 1.8.6
File Edit Sketch Tools Help
sketch
void setup() {
  // put your setup code here, to run once:
}

void loop() {
  // put your main code here, to run repeatedly:
}
```

Слика 5: Шаблон за имплементацију новог Ардуино програма

Поред имплементације нових решења, развојно окружење нуди и готове примере. У примерима је демонстрирано коришћење основних команди за рад са аналогним и дигиталним улазом и излазом, паљење LED сијалице, коришћење дугмета, рад са сензорима. Постоје и примери специфични само за неке ардуино моделе као што су они који демонстрирају рад са тастатуром и мишем.

За прикупљање података потребних за рад прототипа контролне табле коришћена је платформа Arduino UNO приказана на слици 6. Овај модел најзаслужнији је за велику популарност ардуино платформе. Представљен је на сајму посвећеном љубитељима технике у Њујорку који организује познати часопис *Make* и ту је стекао велики број корисника. Постоје три ревизије UNO модела ардуина, а она која се и данас производи је ревизија 3 (R3). Главна разлика у односу на претходне моделе је у типу чипа који конвертује податке са USB порта у серијски облик. Ардуино UNO има чип Atmega8u2 у прве две ревизије, док је за R3 коришћен Atmega16u2. Сви подаци у наставку текста односиће се на трећу ревизију ардуина UNO.



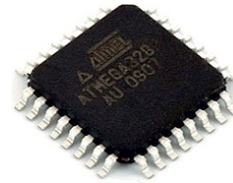
Слика 6: Arduino UNO R3

За напајање ардуина може се користити USB порт или конектор за прикључивање екстерног напајања. Препоручује се напајање са напоном од 5 до 12 волти, а никако преко 20 волти јер може да оштети уређај. Када је уређај прикључен на напајање, на ардуину се пали LED сијалица поред натписа ON. Поред ове сијалице, постоје и TX/RX LED сијалице. Оне пружају визуелно обавештење током серијске комуникације. TX је скраћеница за слање (енг. *transmit*) и ова сијалица се пали када ардуино шаље податке, а RX (енг. *receive*) сијалица се пали приликом преузимања података. Рад ових сијалица може се пратити приликом спуштања новог програма на ардуино. Код спуштен на микроконтролер непрекидно се извршава све док је уређај прикључен на напајање. Уколико је потребно принудно зауставити код, на ардуину постоји ресет дугме. Притискањем овог дугмета прекида се извршавање програма и он се поново покреће испочетка, као да је уређај тек укључен.

Баш као и Raspberry Pi и ардуино поседује пинове опште намене. На њему се налази 14 дигиталних улазно - излазних пинова и 6 пинова за аналогни улаз. Поседује пинове са уземљењем и са сталним напоном од 3.3 и 5 волти. Део дигиталних пинова може да се користи и за слање PWM (енг. *Pulse-Width Modulation*) сигнала. Пинови се налазе на горњој и доњој ивици ардуина. Између њих и испод назива *Arduino UNO* налази се микроконтролер Atmega328. Ардуино UNO R3 има две варијанте које се разликују само по типу кућишта миктоконтролера. У првој је коришћено DIP кућиште, а у другој варијанти користи се SMD кућиште. Оба кућишта су приказана на слици 7.



(I) DIP кућиште



(II) SMD кућиште

Слика 7: Различите варијанте кућишта за микроконтролер на ардуину UNO R3

5 Спецификација захтева и анализа система

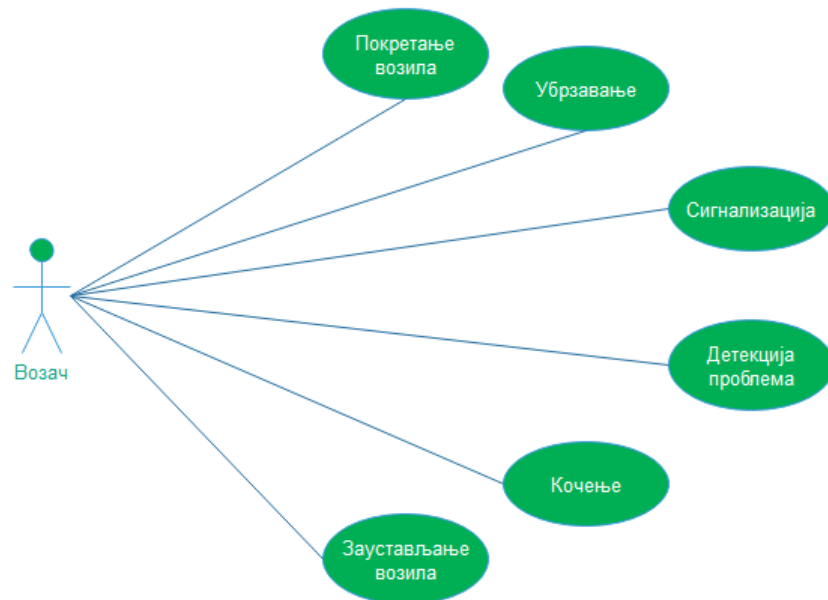
Пре почетка развоја софтвера, потребно је дефинисати захтеве које он мора да испуни. Захтеви се могу односити на начин рада софтвера, перформансе, сигурност или могу бити условљени хардвером на ком ће се софтвер извршавати. Функционални захтеви се односе на начин рада софтвера и морају бити испуњени да би се крајњи производ сматрао употребљивим. Код софтвера за обраду и приказ информација у моторном возилу функционални захтеви су следећи:

- Програм се мора покренути одмах након укључења Raspberry Pi уређаја.
- Непосредно након покретања програма мора се извршити иницијална провера уређаја током које морају бити приказани сви индикатори. Након провере могу остати приказани само индикатори уређаја код којих је детектован проблем и уређаја који су активни.
- На монитору мора постојати приметно и јасно упозорење када је код неког од надгледаних уређаја детектован проблем. Упозорење мора бити упадљиво, али не сме да омета возача у управљању возилом.
- На монитору мора бити приказано обавештење када је неки од надгледаних уређаја активан. Обавештење не сме ометати возача у управљању возилом.
- Током вожње мора бити приказана брзина кретања возила. Брзина може бити приказана са толеранцијом до 5 % више у односу на стварну брзину. Приказана брзина не сме бити мања од стварне брзине.
- Морају бити приказани број обртаја, ниво горива и температура расхладне течности.
- Мора се приказати укупна километража коју је возило прешло и онемогућити измена ове вредности од стране корисника.
- Програм мора да пружи јасан приказ и у условима појачане и смањене светлости у возилу. Јаки сунчеви зраци који падају на монитор не смеју утицати на видљивост елемената. Слика мора бити јасна и у условима ноћне вожње.

Развијани софтвер намењен је извршавању на Raspberry Pi уређају. Пошто је Raspberry Pi рачунар скромних могућности, намећу се ограничења у виду максималне количине ресурса које програм може да користи. Када је RAM меморија у питању, потребе програма не смеју бити веће од 1 GB RAM меморије.

Возило је сложен систем који је изграђен од великог броја засебних система и механизма. Мотор, трансмисија, систем за кочење, систем за снабдевање горивом и расхладни систем су неки од система који омогућују рад возила, а веза путника са њима остварује се преко софтвера на контролној табли. Немају сви путници приступ овом софтверу већ само возач. Његов задатак је да управља возилом поштујући саобраћајне прописе и водећи рачуна о својој безбедности и безбедности осталих учесника у саобраћају. Информације које су му потребне за управљање возилом возач чита са контролне табле.

Управљање возилом се може поделити на неколико етапа: покретање возила, убрзавање, сигнализација осталим учесницима у саобраћају, детекција проблема, кочење и заустављање возила. Свака од ових етапа представља један случај употребе и приказане су на дијаграму на слици 8. Ови случајеви употребе имају само једног учесника и то је возач.



Слика 8: Дијаграм случајева употребе

5.1 Покретање возила

Покретање возила подразумева покретање мотора којем је једна од улога и поизводња енергије за рад осталих уређаја. Један од тих уређаја је и Raspberry

PI на ком се извршава софтвер за контролну таблу и централну конзолу. Након покретања софтвера на контролној табли возачу су доступни статуси уређаја.

Случај употребе: Покретање возила

Актер: Возач

Улаз: Нема

Излаз: Нема

Предуслови: Нема

Постуслови: Упаљен је мотор. Софтвер за контролну таблу је покренут.

Главни ток:

- 1 Возач улази у возило
- 2 Возач подешава ретровизор и седиште
- 3 Возач везује појас
- 4 Возач убацује кључ и пали мотор
- 5 Возач читава статусе уређаја са контролне табле

Алтернативни токови: Нема

5.2 Убрзавање

Када је мотор упаљен, возач може да крене возилом из стања мировања. Покретање возила обавља се низом активности које су дате у опису случаја употребе. Током обављања ових активности возач на контролној табли читава промену брзине, а доступне су му и информације о броју обртаја и броју пређених километара.

Случај употребе: Убрзавање

Актер: Возач

Улаз: Нема

Излаз: Нема

Предуслови: Успешно завршен случај употребе Покретање возила

Постуслови: Нема

Главни ток:

- 1 Возач притиска папучицу за гас и квачило
- 2 Возач помоћу мењача мења степен преноса
- 3 Возач пушта квачило
- 4 Возач притиска папучицу за гас
- 5 Возач читава брзину коју возило постиже на контролној табли
- 6 Возач је достигао жељену брзину и смањује притисак на папучици за гас

Алтернативни токови: Нема

5.3 Сигнализација

Током управљања возилом возач је дужан да обавести остале учеснике у саобраћају о појединим акцијама. Возачу су у те сврхе на располагању контроле за сигнализацију скретања лево или десно и за паљење стоп светла. Ове контроле возач мора да употреби уколико намерава да скрене или да се заустави на коловозу. Приликом скретања постоји механизам који сам искључује сигнално светло након обављене акције, када се волан поравна.

Случај употребе: Сигнализација

Актер: Возач

Улаз: Нема

Израз: Нема

Предуслови: Успешно завршен случај употребе Покретање возила.

Постуслови: Нема

Главни ток:

- 1 Возач пали мигавац за скретање лево и на контролној табли чита обавештење да је мигавац укључен
- 2 Возач скреће лево и чита на контролној табли да је леви мигавац искључен

Алтернативни токови:

2.1. Контролна табла показује да је леви мигавац и даље активан

- 2.1.1. Возач гаси леви мигавац и чита на контролној табли да је мигавац искључен

5.4 Детекција проблема

Током вожње индикатори на контролној табли упозоравају возача на детектоване проблеме. Возач мора да одреагује на та упозорења како не би дошло до озбиљнијих проблема. Упозорења се могу односити на: положај ручне кочице, истрошеност кочионих плочица, ниво кочионе течности, стање акумулатора, притисак уља, проблем са радом мотора или температуру расхладне течности. Зависно од тога који је проблем у питању, возач одлази до најближег сервиса, зауставља возило и сипа расхладну течност или уље. Уколико је активан индикатор положаја ручне кочице, возач мора спустити ручну како би могао да започне са вожњом. У супротном би дошло до оштећења. Дијаграм активности на слици 9 показује на који начин возач треба да реагује на свако од поменутих упозорења.

Случај употребе: Детекција проблема

Актер: Возач

Улаз: Нема

Излаз: Нема

Предуслови: Успешно завршен случај употребе Покретање возила

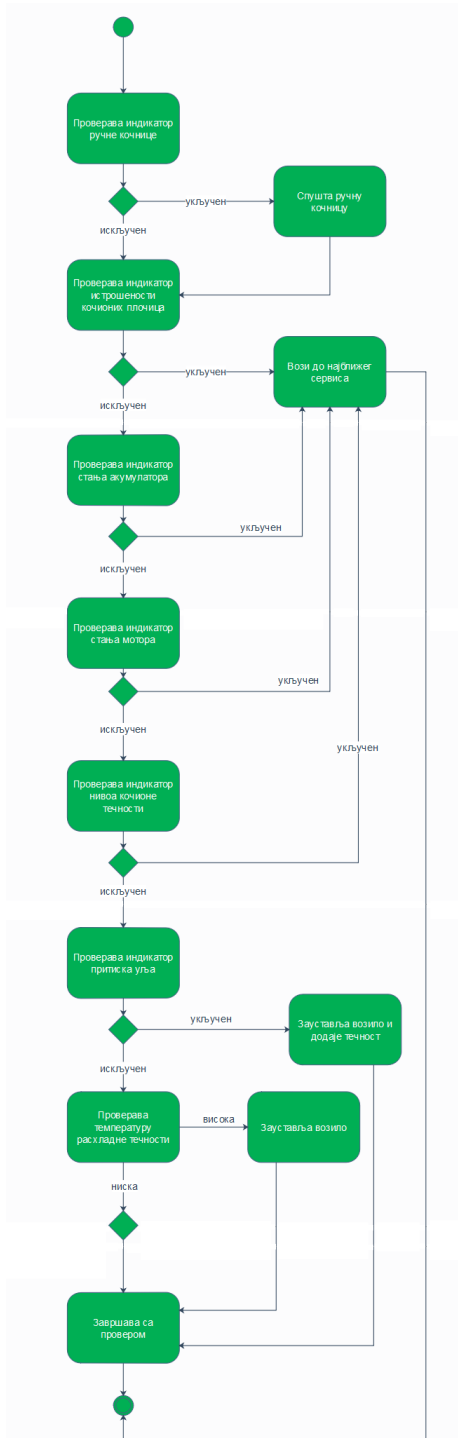
Постуслови: Нема

Главни ток:

- 1 Возач на контролној табли проверава индикатор положаја ручне кочице
- 2 Возач на контролној табли проверава индикатор истрошености кочионих плочица
- 3 Возач читава статус акумулатора са контролне табле
- 4 Возач читава статус мотора са контролне табле
- 5 Возач на контролној табли проверава индикатор нивоа кочионе течности
- 6 Возач на контролној табли проверава индикатор притиска уља
- 7 Возач читава температуру расхладне течности

Алтернативни токови:

- 1.1. Индикатор показује да је ручна кочица подигнута
 - 1.1.1. Возач спушта ручну кочицу
- 2.1. Индикатор истрошености кочионих плочица је активан
 - 2.1.1. Возач наставља вожњу до најближег сервиса
- 3.1. Индикатор стања акумулатора је укључен
 - 3.1.1. Возач наставља вожњу до најближег сервиса
- 4.1. Индикатор стања мотора је активан
 - 4.1.1. Возач наставља вожњу до најближег сервиса
- 5.1. Индикатор нивоа кочионе течности је активан
 - 5.1.1. Возач наставља вожњу до најближег сервиса
- 6.1. Индикатор притиска уља је активан
 - 6.1.1. Возач зауставља возило и сипа уље
- 7.1. Температура расхладне течности је превисока
 - 7.1.1. Возач зауставља возило и подиже хаубу



Слика 9: Дијаграм активности за случај употребе: детекција проблема

5.5 Кочење

Током вожње возач смањује брзину возила и зауставља возило коришћењем кочница. У процесу кочења возач притиска папучицу и прати промену брзине на контролној табли.

Случај употребе: Кочење

Актер: Возач

Улаз: Нема

Изназ: Нема

Предуслови: Успешно завршен случај употребе Убрзавање.

Постуслови: Нема

Главни ток:

- 1 Возач управља возилом које се креће брзином већом од нуле
- 2 Возач притиска кочницу
- 3 Возач на контролној табли читава смањење брзине возила

Алтернативни токови: Нема

5.6 Заустављање возила

Заустављање возила реализује се постепеним смањењем брзине све док се возило не заустави. Овај случај употребе се реализује када је црвено светло на семафору, али и приликом паркирања возила када се након смањења брзине подиже ручна кочница и искључује мотор. Ручна кочница се активира ручицом поред возачевог седишта. На контролној табли налази се индикатор који показује да ли је ручна кочница подигнута.

Случај употребе: Заустављање возила

Актер: Возач

Улаз: Нема

Изназ: Нема

Предуслови: Возило је покренуто и креће се.

Постуслови: Возило је заустављено и мотор је искључен.

Главни ток:

- 1 Возач извршава случај употребе Кочење
- 2 Возач читава на контролној табли да је брзина возила пала на 0 km/h
- 3 Возач подиже ручну кочницу
- 4 Возач окреће кључ и гаси возило

Алтернативни токови: Нема

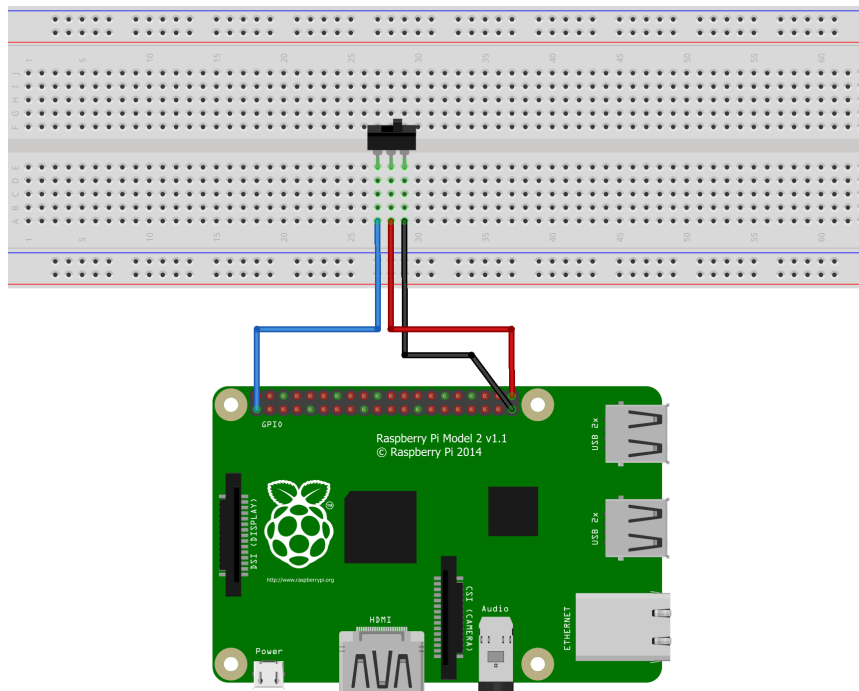
6 Дизајн решења

Потребно је да развијени софтвер прикупи информације и прикаже их исцртавањем објеката на екрану. У поглављу 2 је за сваки елемент контролне табле описано на који начин ће тај елемент добити своју вредност. Велика група елемената вредност добија преко сензора. Raspberry Pi може остварити везу са сензорима преко пинова опште намене и тако детектовати јављање догађаја. Зависно од тога да ли се догађај јавио, на екрану се исцртава одређени индикатор. За елементе чија вредност не може да се добије преко сензора, као што су брзина и број обртаја, погодније је користити ардуино. Ардуино не може сам да детектује све ове вредности и зато је потребно користити додатне уређаје.

Приликом имплементације прототипа контролне табле све догађаје потребно је симулирати. Сензор се може заменити струјним колом са прекидачем. Померањем дугмета прекидача и пропуштањем струје до пина симулира се слање сигнала. Овако се симулирају догађаји приказани у табели 1. У другој колони табеле је приказано који пин се користи за детектовање ког догађаја. Пинови су дати ВСМ бројем. Повезивање пина и прекидача приказано је на слици 10 на примеру индикатора за дуго светло. Прекидач има три пина која су повезана са извором напајања, уземљењем и пином опште намене на Raspberry Pi уређају ком се шаље сигнал. Након што се детектује сигнал исцртава се индикатор за дуга светла. Овај индикатор је константно присутан на монитору све док се померањем прекидача не онемогући доток струје до пина. Није приказ свих индикатора константан. Индикатори упозорења и показивачи правца трепере када су активни. У табели 1 у колони *приказ* индикаторима су додељене вредности **треперење** и **константно**.

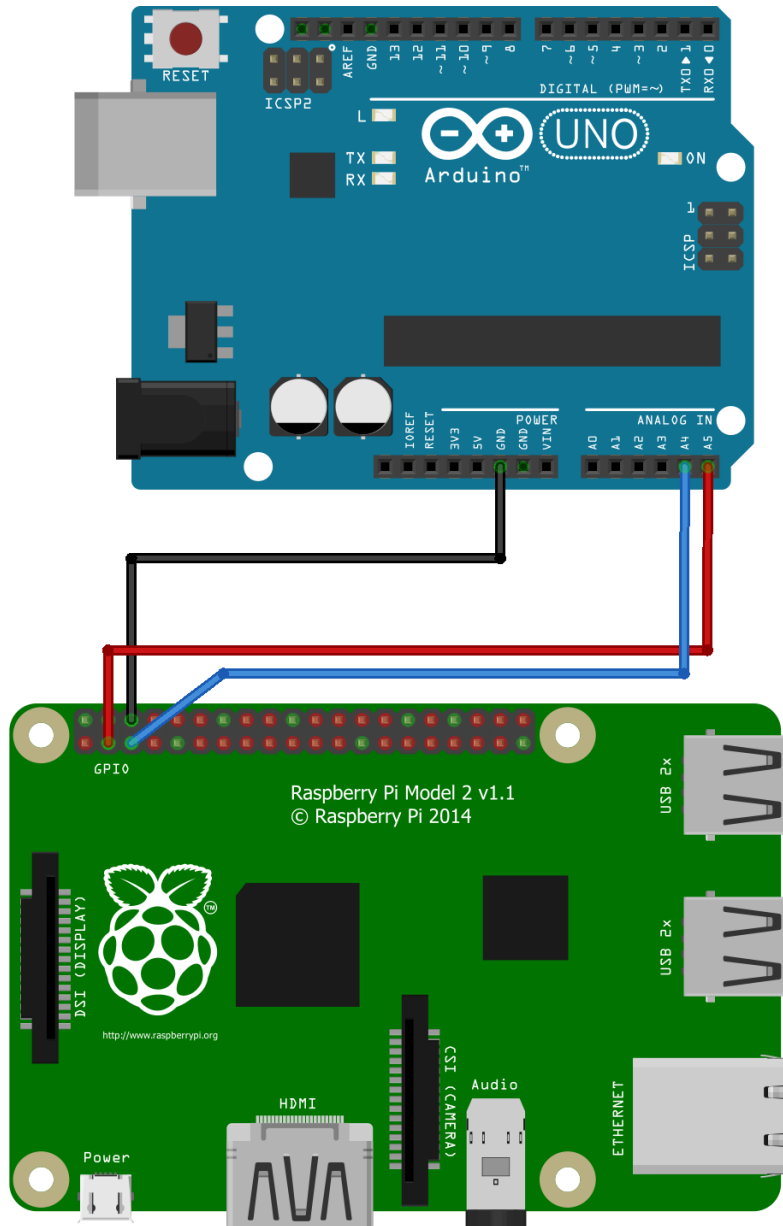
Табела 1: Мапирање пинова опште намене

индикатор	пин	приказ
положај ручне кочнице	19	треперење
истрошеност кочионих плочица	13	треперење
ниво кочионе течности	12	треперење
провера мотора	25	треперење
стање акумулатора	6	треперење
притисак уља	5	треперење
светла за маглу	16	константно
дуга светла	21	константно
леви мигавац	26	треперење
десни мигавац	20	треперење



Слика 10: Повезивање прекидача за симулацију сигнала дугог светла

За симулацију података који се шаљу са Ардуина погодније је софтверско решење. Посматране вредности се могу ажурирати у циклусу који се константно извршава на микроконтролеру. Размена података између ардуина и Raspberry Pi-а може се реализовати неким од протокола за серијску комуникацију. Оба уређаја подржавају и SPI и I²C протокол. Пошто је за I²C протокол потребно мање комуникационих канала, а самим тим и мањи број пинова, овај протокол биће коришћен у имплементацији прототипа. Пинови намењени комуникацији по I²C протоколу код Raspberry Pi-а су физички пинови 3 и 5. Пин 3 шаље податке, а пин 5 сигнал часовника. На ардуину UNO пин A4 се користи за податке, а пин A5 за сигнал часовника. Поред ова два пина потребно је повезати ардуино са пином за уземљење на Raspberry Pi-у. Схема која приказује како су повезани ови уређаји дата је на слици 11. Подаци који се шаљу на овај начин су брзина, километража, број обртаја, ниво горива и температура расхладне течности.



Слика 11: Повезивање ардуина и Raspberry Pi-a

7 Имплементација решења

Развијени софтвер се извршава на Raspbian Light оперативном систему, на верзији 8 познатијој под називом *Jessie*. На оперативном систему је инсталирано LXDE (енг. *Lightweight X11 Desktop Environment*) десктоп окружење. Целокупан код написан је у С програмском језику, по стандарду C99. За графички део рада коришћена је OpenGL библиотека. Приступ пиновима опште намене реализован је преко *sysfs* интерфејса, а за имплементацију комуникације по I^2C протоколу коришћене су библиотеке *i2c* и *i2c-dev*. Велики део дизајна чине текстуре. Парсирање датотека коришћених за текстуре реализовано је помоћу библиотеке *libpng-1.6.34*.

Неке од операција над пиновима опште намене може да изврши само корисник са максималним администраторским правима. Зато је неопходно да извршни програм буде покренут од његове стране. Пошто је један од захтева да се развијени софтвер покреће аутоматски након паљења уређаја, направљена је скрипта која је додата у листу за аутоматско покретање након покретања графичког окружења. Датотека у којој се ова листа налази је `autostart` и налази се на локацији `/home/pi/.config/lxsession/LXDE-pi/autostart`. Код на листингу 1 показује како се додаје нови елемент у листу за покретање, а садржај скрипта дат је у листингу 2. Пошто је потребно неколико секунди након покретања графичког окружења да би се слика приказала на екрану, на почетку скрипта позива се функција `sleep` која чека да прођу 2 секунде пре него што се настави извршавање команди. Након тога мења се локација радног директоријума, прелази се у директоријум у ком се налази извршни програм и програм се покреће.

```
1 @sh /home/pi/dashboard_start.sh
```

Листинг 1: Додавање скрипта у листу за покретање у `autostart` датотеци

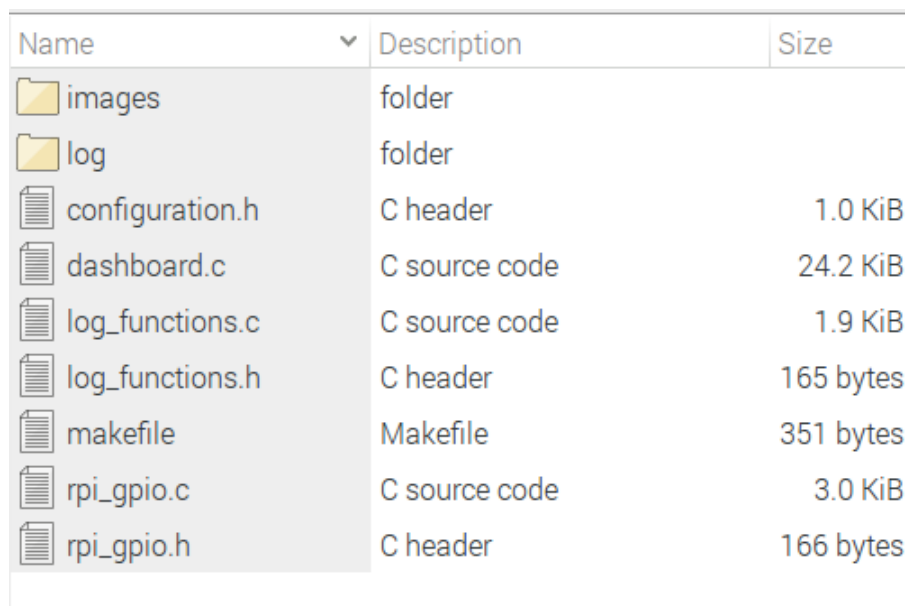
```
1 #!/usr/bin/sh
2 sleep 2
3 cd /home/pi/kontrolnaTabla
4 ./dashboard
```

Листинг 2: Садржај скрипта `dashboard_start.sh`

Развијени код је физички подељен у неколико датотека, али се могу уочити и логичке целине. На слици 12 приказана је структура директоријума у ком се изворни код налази. Свака датотека може се доделити одређеној логичкој целини осим датотеке `dashboard.c`. То је датотека са главном логиком програма и у њој се

налазе функције различите намене које раде са глобалним променљивама. Опис имплементације биће дат кроз следеће целине:

- конфигурација
- упис у дневник
- обрада слика
- контрола GPIO пинова
- комуникација по I^2C протоколу
- исцртавање елемената



Name	Description	Size
images	folder	
log	folder	
configuration.h	C header	1.0 KiB
dashboard.c	C source code	24.2 KiB
log_functions.c	C source code	1.9 KiB
log_functions.h	C header	165 bytes
makefile	Makefile	351 bytes
rpi_gpio.c	C source code	3.0 KiB
rpi_gpio.h	C header	166 bytes

Слика 12: Структура директоријума са изворним кодом

7.1 Конфигурација

Конфигурација обухвата дефинисање путања до слика коришћених за текстуре и дефинисање вредности интервала за часовник. Ови подаци су смештени у датотеци *configuration.h*, а слике на које се реферише су у директоријуму *images*. Подржани формат слика за текстуре је *png*.

Конфигурација часовника се односи на дефинисање вредности два макроя `TIMER_ID` и `TIMER_INTERVAL`. Први макро је идентификатор часовника, а други представља временски интервал након кога ће се функција `on_timer()` позвати поново. У овој функцији читавају се подаци које треба приказати и

иницира се поновно исцртавање садржаја екрана. Интервал позива функције `on_timer()` је од велике важности за апликацију. Премала вредност може довести до брзог мењања садржаја на екрану до те мере да садржај постане нечитљив, док велика вредност узрокује кашњење у обавештавању корисника о статусу возила. Оваква кашњења могу да узрокују велике проблеме у систему као што је моторно возило где је битно да се детекција и приказ обаве непосредно након јављања догађаја. Интервал након ког се врши освежавање приказа у развијеном софтверу је 750 милисекунди, што износи 80 исцртавања садржаја екрана у минути.

7.2 Упис у дневник

Код софтвера за контролну таблу, заустављање програма у случају грешке није опција. Зато се током извршавања програма поруке о грешкама бележе у дневник-датотеку. Ако је податак који треба приказати недоступан, на екрану се исцртава знак са слике 13 који упозорава корисника да нису сви подаци успешно прочитани. Уколико се овај индикатор не искључи након неколико секунди, најсигурније је прекинути вожњу. У дневник-датотеци остаје забележено који податак је недоступан и након вожње може се извршити детекција и отклањање квара. Подаци у дневнику се памте највише 30 дана. Датотеке чија је величина већа од 10 MB се бришу при првом следећем покретању програма.



Слика 13: Знак упозорења возачу да поједини подаци нису успешно прочитани

Имплементирана су три нивоа дневничких података и то за информације - ниво 0, упозорења - ниво 1 и грешке - ниво 2. Код нивоа 1 пре поруке уписује се кључна реч **WARNING**, а кључна реч **ERROR** код порука нивоа 2. Дефиниције функција за рад са дневником су у датотеци `log_functions.c`, а како би се упис у дневник могао извршити успешно директоријум `log` мора бити креиран у истом директоријуму у ком је и извршна датотека. Датотеке са дневницима носе назив `dashboardddmmyyy.log` где `ddmmyyy` означава датум креирања датотеке. Дневник-датотека се креира приликом покретања програма. Уколико се програм покрене 03.05.2018. године пре поноћи и непрекидно се извршава до

04.05.2018. године до 00:30 h, целокупни дневнички подаци налазиће се у датотеци *dashboard03052018.log*. Ово не представља проблем у тумачењу података јер сваки ред датотеке почиње датумом и временом уписа.

Декларације функција које учествују у имплементацији рада са дневником дате су кодом у листингу 3. Функција `clear_old_logs()` приказана у листингу 5 се позива на почетку програма, пре иницијализације података. Као аргумент функцији се прослеђује путања до директоријума са дневницима. Позивом функције `system` бришу се стари или превелики дневници. Мана функције `system` је што се њено извршавање не може прекинути, а програм који ју је позвао мора сачекати да заврши. Како услед неке грешке извршавање ове функције не би узроковало паузирање рада програма, решено је да се функција `clear_old_logs()` извршава у посебној нити. Листинг 4 показује како се нит креира. Позив функције `pthread_detach()` одваја нит од осталих и када она заврши са извршавањем њени ресурси се аутоматски ослобађају.

```
1 int create_log_file(char* path);
2
3 void log_data(int level, const char* format, ...);
4
5 void* clear_old_logs(void* path);
6
7 void close_log_file();
```

Листинг 3: Функције за рад са дневник-датотекама

```
1 pthread_t thread_id;
2
3 ret = pthread_create(&thread_id, NULL, clear_old_logs, "/home/pi/
  kontrolnaTabla/log/");
4
5 if (ret == 0)
6     pthread_detach(thread_id);
7
8 create_log_file("/home/pi/kontrolnaTabla/log/");
```

Листинг 4: Део кода из функције `initialize()` за креирање нити

```
1 void* clear_old_logs(void* path)
2 {
3     char command[256];
4
5     sprintf(command, "find %s -type f \\( -mtime +30 -or -size +10M \\) -
  delete", (char*) path);
```

```

6
7   system (command) ;
8
9   log_data (0, "logs DELETED\n");
10
11  return NULL;
12 }

```

Листинг 5: Функција за брисање дневник-датотека

Наредна функција која се позива је `create_log_file()`. Она креира дневник-датотеку са данашњим датумом. Уколико је датотека успешно креиран функција враћа 0, а у случају грешке -1. Ако се јави грешка показивач на FILE структуру имаће вредност NULL. Функција `log_data()` задужена је за упис у дневник-датотеку. На почетку у функцији се проверава да ли је показивач на FILE структуру NULL и у том случају остатак функције се не извршава. Провера вредности показивача на FILE структуру имплементирана је и у функцији за затварање датотеке `close_log_file()`.

7.3 Обрада слика

Библиотека *OpenGL* омогућава цртање тачке, линије и полигона. Сви остали објекти граде се од њих. Често је потребно приказати сложен симбол као што су они приказани на слици 14. Једноставно решење за приказ ових симбола је коришћење текстуре. Текстура у *OpenGL*-у је објекат који садржи једну или више слика истог формата и покрива површину полигона. Формат слика за текстуре који развијени софтвер подржава је *png*.



Слика 14: Симболи за температуру расхладне течности и ниво горива

Приликом креирања текстуре неопходни су подаци о висини, ширини и пикселима. Обрада слике подразумева парсирање датотеке како би се ови подаци преузели. Имплементирана функција за парсирање *png* датотеке је `init_png_light()` и налази се у датотеци *dashboard.c*. Функцији се као аргументи прослеђују показивач на низ текстура, индекс текстуре која се иницијализује, име *png* датотеке од које ће текстура бити креирана и формат. Аргумент формат односи се на

број компонената боје у текстури. Подржани су формати дефинисани макроима `GL_RGB` и `GL_RGBA`.

7.4 Контрола GPIO пинова

Пинови опште намене описани су у поглављу 3. За рад са њима коришћен је *sysfs* интерфејс. Имплементирани су четири функције које се налазе у датотеци *gpi_gpio.c*. То су функције за извоз и уклањање пина, читање вредности и постављање правца. За сваки пин потребно је прво урадити извоз. Овим се креира директоријум са датотекама за контролу пина. Новонастали директоријум носи назив *gpioK* где је *K* BCM број пина. У директоријуму се налази датотека *value* чији садржај је 0 или 1, зависно од тога да ли је пин повезан на 0 V или на 3.3 V. Имплементација функције која приступа овој датотеци и чита њен садржај приказана је на листингу 6. Као аргумент функцији се прослеђује BCM број пина, а она враћа 0 или 1 зависно од тога шта је прочитано из датотеке. У случају грешке функција враћа нулу.

Друга значајна датотека је *direction* чији садржај одређује смер пина и може имати садржај *in* или *out*. Функција којом се задаје смер зове се `set_direction()` и њена имплементација се састоји од уписа кључне речи прослеђене као аргумент у датотеку. Ова функција се позива на почетку програма, приликом иницијализације података. Сличне су и имплементације функција `export_pin()` и `unexport_pin()` с тиме што се код њих у датотеку уписује број пина за који је потребно извршити извоз, тј. који је потребно уклонити. Ове функције враћају нулу у случају успеха, а -1 у случају грешке. Могуће грешке су прослеђени број пина који не припада опсегу [0, 40] или грешка приликом уписа у датотеку.

```
1 int get_value(int gpio_num)
2 {
3     char file_name[40], value[2];
4
5     if (gpio_num < 0 || gpio_num > 40)
6     {
7         log_data(2, "BCM number %d is not supported\n", gpio_num);
8         return 0;
9     }
10
11     sprintf(file_name, "/sys/class/gpio/gpio%d/value", gpio_num);
12     int fd = open(file_name, O_RDONLY);
13
14     if (fd == -1)
15     {
16         log_data(2, "failed to open ~%s~ file!\n", file_name);
```



```

17     return 0;
18 }
19
20 int r = read(fd, value, 2); //0 ili 1
21
22 if (r < 1)
23 {
24     log_data(2, "read in get_value(%d) returned %d\n", gpio_num, r);
25     close(fd);
26     return 0;
27 }
28
29 value[1] = '\0';
30
31 close(fd);
32
33 int ivalue = atoi(value);
34
35 if (ivalue != 0 && ivalue != 1)
36     ivalue = 0;
37
38 return ivalue;
39 }

```

Листинг 6: Код за читање вредности пина

7.5 Комуникација по I^2C протоколу

У поглављу 6 описано је како су ардуино и Raspberry Pi повезани. Код описан у листингу 7 извршава се на ардуину. При иницијализацији, тј. у `setup()` функцији се поставља адреса за споредни уређај и функција која ће се позивати када главни уређај пошаље захтев. То је функција `sendData()` и у њој се форматира и шаље стринг са поруком. Стринг је форматирани тако да делимитер | одваја сегменте. Сваки сегмент састоји се од тага дужине три карактера који означава која мерена величина је у питању и бројне вредности. Вредности мерених величина се сваке секунде ажурирају у `loop()` функцији .

```

1 #include <Wire.h> //biblioteka za i2c komunikaciju
2 #include <stdio.h>
3
4 #define SLAVE_ADDRESS 0x40 //port na koji sam prikacila zice
5
6 uint8_t tmp = 0; //temperatura
7 uint8_t fue = 0; //gorivo

```

```

8 uint8_t tac = 0; //tahometar
9 uint8_t spd = 0; //brzina
10 int32_t odm = 0; //odometar
11
12 void setup() {
13   Wire.begin(SLAVE_ADDRESS);
14   Wire.onRequest(sendData); //funkcija koja se poziva prilikom zahteva
15 }
16
17 void loop() {
18   tmp++;
19   if (tmp == 101) tmp = 0;
20
21   fue++;
22   if (fue == 101) fue = 0;
23
24   tac++;
25   if (tac == 8) tac = 0;
26
27   spd++;
28   if (spd == 250) spd = 0;
29
30   odm++;
31   if (odm == 1000000) odm = 0;
32
33
34   delay(1000);
35 }
36
37 void sendData() {
38   char data[64];
39
40   sprintf(data, "TMP%d|FUE%d|TAC%d|SPD%d|ODM%d|", tmp, fue, tac, spd,
41             odm);
42   Wire.write(data);
43 }

```

Листинг 7: Код на Ардуину

На Raspberry Pi уређају се пре сваког исцртавања садржаја на екрану позива функција `read_i2c_data()` која је представљена у листингу 9. У овој функцији имплементирана је комуникација по I^2C протоколу коришћењем датотеке `i2c-1`. Помоћу системског позива `ioctl` шаље се захтев споредном уређају и уколико је позив био успешан чита се одговор. Уколико се догоди грешка и подаци се не преузму са ардуина, функција `read_i2c_data()` враћа -1. Ако су подаци успешно прочитани, позива се функција `update_i2c_dependent_values()` која парсира

string прочитан са ардуина и ажурира вредности променљивих.

Функције за I²C комуникацију зову се приликом иницијализације података и у `on_timer()` функцији датој у листингу 8. Ова функција је одговорна за ажурирање променљивих у којима се чувају посматране вредности и за инцирање поновног исцртавања садржаја на екрану. Променљива `animation_ongoing` чува информацију о томе да ли се индикатори који трепере исцртавају тако што се њена вредност додељује статусу индикатора, ако је активан. Индикатор се сматра активним ако пин задужен за праћење његове вредности детектује напон од 3.3 V. На крају функције неопходно је дефинисати временски интервал након ког се функција поново извршава. То се постиже позивом `glutTimerFunc()`.

```
1 static void on_timer(int value)
2 {
3     int i;
4
5     if(value != 0)
6         return;
7
8     animation_ongoing = (animation_ongoing + 1) % 2; //kontrolise
9         treperenje indikatora
10
11     for(i = 0; i < 10; i++)
12     {
13         lights[i].value = get_value(lights[i].gpio); //RPi:
14
15         if (i == 0 || i == 7)
16             if (lights[i].value > 0)
17                 lights[i].status = 1;
18             else
19                 lights[i].status = 0;
20         else
21             if (lights[i].value > 0)
22                 lights[i].status = animation_ongoing; //1 - upali, 0 - ugasi
23             else
24                 lights[i].status = 0;
25     }
26
27     if (read_i2c_data() > 0)
28         update_i2c_dependent_values();
29
30     glutPostRedisplay(); //forsira ponovno iscrtavanje prozora
31     glutTimerFunc(TIMER_INTERVAL, on_timer, TIMER_ID);
32 }
```

Листинг 8: Функција `on_timer()`

```

1 static int read_i2c_data()
2 {
3     int i2c_file;
4
5     i2c_file = open("/dev/i2c-1", O_RDWR);
6
7     if (i2c_file < 0)
8     {
9         logData(logFile, 2, "trying to open i2c communication files\n");
10        data_status = 1;
11        return -1;
12    }
13
14
15    int addr = 0x40;
16
17    if (ioctl(i2c_file, I2C_SLAVE, addr) < 0)
18    {
19        logData(logFile, 2, " trying to access slave device\n");
20        data_status = 1;
21        return -1;
22    }
23
24    unsigned int length = 63;
25
26
27    int k = read(i2c_file, i2c_buffer, length) ;
28
29    if(k == -1)
30    {
31        logData(logFile, 0, "Read error\n");
32        close(i2c_file);
33        data_status = 1;
34        return -1;
35    }
36
37    logData(logFile, 0, "Read data: %s (%d)\n", i2c_buffer, k);
38
39    close(i2c_file);
40
41    return 1;
42 }

```

Листинг 9: Код на Raspberry Pi уређају за комуникацију са Ардуином

7.6 Исцртавање елемената

Целокупан код за исцртавање елемената на контролној табли смештен је у датотеку `dashboard.c`. У овој датотеци налази се `main` функција у којој се иницијализује `GLUT`, позива функција за `OpenGL` иницијализацију, региструју функције повратног позива (енг. *callback*), подешава осветљење и започиње обрада догађаја. Поред `OpenGL` иницијализације у функцији `initialize` креира се дневник-датотека, извозе се пинови опште намене, поставља се њихов смер и контактира се ардуино ради иницијализације променљивих. Уколико веза са ардуином није успостављена, одговарајућим променљивама се додељује вредност 0.

Декларације функција задужених за приказ објеката на екрану дате су на листингу 10. Разликују се две групе функција: функције повратног позива и функције за исцртавање објеката. Функције повратног позива се извршавају након одређеног догађаја и то су функције `reshape()` и `display()`. Функција `reshape()` позива се приликом промене димензија прозора и у њој се подешава област која ће бити приказана на екрану. Када је потребно исцртати садржај прозора позива се функција `display()`. Ова функција задужена је за позиционирање камере и позиве функција које цртају елементе сцене. Сегмент функције за исцртавање објеката дат је у листингу 11. На крају функције `display()` проверава се да ли је ово први улазак у функцију и ако је тако позива се функција `on_timer()` по први пут. Приликом овог позива чека се две секунде, а не 750 милисекунди као у наставку извршавања програма. Разлог за ово је захтев да након покретања програма буду приказани сви индикатори и да се они по потреби могу угасити тек након провере уређаја. Изглед екрана током иницијалне провере дат је на слици 15.

```
1 static void reshape(int width, int height);
2 static void display(void);
3 static void on_timer(int value);
4
5 static void draw_light(GLuint* texture, int k, float start_x, float
   start_y, float w, float h);
6
7 static void draw_side_gauge(float start_x, float start_y, unsigned int
   proc, char color, int texture);
8 static void draw_center_of_scene(float start_x, float start_y);
```

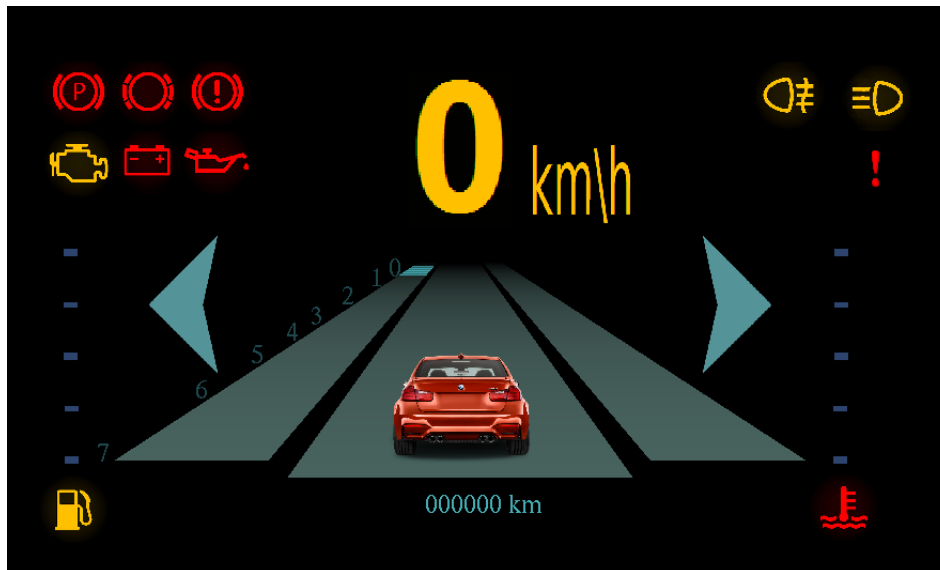
Листинг 10: Функције за контролу приказа на екрану

```

1 draw_center_of_scene(0, 0);
2 draw_side_gauge(-2.25, -1.4, fuel, 'y', 13);
3 draw_side_gauge(2.2, -1.4, collant_tmp, 'r', 12);
4
5 for(i = 0; i < 6; i++)
6     if (lights[i+1].status == 1)
7         draw_light(images, i+1, -2.4 + (i%3) * 0.4, i > 2 ? 0.4 : 0.8, 0.4,
8             0.4);
9
10 if (lights[0].status == 1) draw_light(images, 0, 1.7, 0.8, 0.4, 0.4);
11 if (lights[7].status == 1) draw_light(images, 7, 2.2, 0.8, 0.4, 0.4);
12
13 if (data_status == 1)
14 {
15     draw_light(images, 14, 2.3, 0.45, 0.2, 0.25);
16     data_status = 0;
17 }
18
19 glMaterialfv(GL_FRONT, GL_AMBIENT, cITile);
20 glMaterialfv(GL_FRONT, GL_DIFFUSE, cITile);
21 glRasterPos2f(-0.2, -1.4);
22 glLineWidth(7);
23
24 char d[10];
25 sprintf(d, "%.6d km", odom_val);
26 glutBitmapString(GLUT_BITMAP_TIMES_ROMAN_24, (const unsigned char*)d);
27
28 if (speed >= 0 )
29     draw_speed(0, 0.7, speed);
30
31 draw_light(images, 10, -0.6, -1.2, 1.2, 0.8);
32
33 if (lights[8].status == 1) draw_light(images, 8, -1.8, -0.6, 0.4, 0.8);
34 if (lights[9].status == 1) draw_light(images, 9, 1.4, -0.6, 0.4, 0.8);
35
36 glutSwapBuffers();
37
38 if (initialization_ongoing == 1)
39 {
40     glutTimerFunc(2000, on_timer, TIMER_ID);
41     initialization_ongoing = 0;
42     log_data(0, "Started on_timer function\n");
43 }

```

Листинг 11: Део функције display



Слика 15: Изглед екрана током иницијалне провере

Приликом исцртавања објеката могуће је уочити неколико група:

- објекти који чине део дизајна и не дају никакве информације кориснику
- индикатори
- скале које приказују вредност мерене величине
- бројне вредности мерених величина

У функцији `draw_center_of_scene()` исцртавају се објекти који чине део дизајна и скала на којој се приказује вредност броја обртаја. Код приказан на листингу 12 приказује део ове функције којим се исцртавају бројеви поред скале и плочице које означавају ком броју на скали одговара број обртаја. Бројеви се исцртавају помоћу битмапе. Скала за приказ вредности броја обртаја протеже се по z оси, а број плочица није исти на сваком подеоку и расте како се скала удаљава од прве равни одсецања, тј. више плочица налази се између мањих вредности на скали. Укупан број плочица које се могу приказати је 16 и оне се исцртавају почевши од плочице која је најдаље по z оси па до плочице која се налази поред вредности која одговара тренутном броју обртаја. Вредност до које треба исцртати плочице чува се у променљивој *granica* и рачуна се по формули $7 - \text{tacho_val}$ ако је број обртаја већи од 2, а $3 * (4 - \text{tacho_val})$ иначе. Променљива *tacho_val* чува вредност броја обртаја.

```

1 glLineWidth(7);
2 glMaterialfv(GL_FRONT, GL_AMBIENT, cITile);
3 for (int i = 0; i < 8; i++)
4 {
5     char index[2];
6     sprintf(index, "%d", 7-i);
7     if (7 - i > 2)
8         glRasterPos3f(-2.08 - 0.02*i, -1.09, 0.1-4*i);
9     else
10        glRasterPos3f(-2.09-0.03*i, -1.09, 0.1-8*i + 4 * (7-i+2));
11
12    glutBitmapString(GLUT_BITMAP_TIMES_ROMAN_24, (const unsigned char*)
13        index);
14 }
15 //levi
16 glMaterialfv(GL_FRONT, GL_AMBIENT, cITile);
17 glMaterialfv(GL_FRONT, GL_DIFFUSE, cITile);
18
19 int granica = tacho_val > 2 ? 7 - tacho_val : 3 * (4 - tacho_val);
20
21 glBegin(GL_QUADS);
22     for(j=16; j >= granica; j--)
23     {
24         if (j == 1)
25         {
26             glMaterialfv(GL_FRONT, GL_AMBIENT, clRed);
27             glMaterialfv(GL_FRONT, GL_DIFFUSE, clRed);
28         }
29
30         glNormal3f(0, 1, 0);
31         glVertex3f(-1.1, -1.09, 0.1-4*j);
32         glVertex3f(-2.0, -1.09, 0.1-4*j);
33         glNormal3f(0, -1, 0);
34         glVertex3f(-2.0, -1.09, 0.1-4*(j+1));
35         glVertex3f(-1.1, -1.09, 0.1-4*(j+1));
36     }
37 glEnd();

```

Листинг 12: Део функције draw_center_of_scene()

Контролна табла садржи још две скале, за приказ температуре расхладне течности и нивоа горива у резервоару. Ове скале исцртавају се функцијом draw_side_gauge() која као аргументе очекује x и y координату на којој ће се налазити доњи леви угао скале, вредност коју треба приказати на скали, боју линије која означава ниво на скали и број текстуре која се користи за цртање симбола. Цртање симбола реализовано је помоћу функције draw_light() којој се

прослеђују низ текстура, индекс текстуре која се користи, x и y координате доњег левог угла, ширина и висина. Символ се исцртава као правоугаони полигон на ком се налази задата текстура. Помоћу ове функције цртају се и индикатори, али они се исцртавају само ако је преко пинова опште намене детектовано јављање догађаја на који указују. За сваки индикатор чувају се информације у структури `Light` приказаној листингом 13. Поље `gpio` означава ВСМ број пина са ког се очитава вредност индикатора која се потом смешта у поље `value`. Поље `status` означава да ли при наредном исцртавању сцене треба исцртати и овај индикатор. Ова вредност је увек 1 код индикатора који су константно упаљени када је догађај на који се односе детектован. То су индикатори за дуга светла и светла за маглу. Код индикатора који трепере када се догађај детектује, као што је индикатор правца, вредност поља `status` се мења између исцртавања и уколико не треба исцртати индикатор има вредност 0.

```
1 typedef struct Light{
2     unsigned int gpio;
3     unsigned int status;
4     unsigned int value;
5 }Light;
```

Листинг 13: Дефиниција структуре `Light`

Бројним вредностима су на контролној табли приказани одометар и брзина. Одометар се исцртава помоћу битмапе, а брзина помоћу текстуре. Разлог зашто и за брзину није коришћена битмапа је тај да је ово једна од најбитнијих вредности на контролној табли и мора бити упадљиво приказана. Исцртава се помоћу функције `draw_speed()` којој се прослеђују x и y координата центра полигона и брзина. Функција је дата кодом у листингу 14.

```
1 int draw_speed(float x_center, float y_center, int speed)
2 {
3     int digits_num = 0;
4     int tmp = speed;
5     int speed_digits[4];
6
7     if (speed < 0)
8         return digits_num;
9
10    if (tmp == 0)
11    {
12        speed_digits[0] = 0;
13        digits_num = 1;
14    }
15    else
```

```

16 while(tmp!=0 && digits_num < 4)
17 {
18     speed_digits[digits_num] = tmp%10;//popunjavam obrnutim redosledom
19     digits_num++;
20     tmp = tmp/10;
21 }
22
23 float x_kmh = 0.3;
24 float start_x=0.3, start_y=y_center - 0.45;
25 for(int i = 0; i < digits_num; i++)
26 {
27
28     if (digits_num > 1)
29         start_x = digits_num % 2 == 0? x_center - 0.6 * i : 0.3 - i * 0.6;
30     else
31         start_x = x_center - 0.3;
32
33     if (i == 0)
34         x_kmh = start_x + 0.7;
35
36     draw_light(digits, speed_digits[i], start_x, start_y, 0.6, 0.9);
37 }
38
39 draw_light(images, 11, x_kmh, start_y, 0.6, 0.5);
40
41 return digits_num;
42 }

```

Листинг 14: Испртавање брзине

8 Тестирање

Тестирање је реализовано у неколико фаза од којих свака има различит циљ. У првој фази тестирано је испртавање елемената. Тест се обавља помоћу тастатуре и прати се приказ на екрану. Други тест има за циљ проверу детектовања догађаја које шаљу пинови опште намене. Трећим тестом проверава се комуникација са ардуином по I^2C протоколу.

8.1 Тестирање графичког корисничког интерфејса

Тестирање графичког корисничког интерфејса се спроводи како би се утврдило да се сви елементи испртавају на предвђен начин. Код софтвера за обраду и приказ информација у моторном возилу то значи да елементи на екрану треба

да задовоље захтеве из поглавља 5. Овим тестовима проверава се да ли су елементи јасно приказани, које су максималне вредности које се могу приказати, шта се дешава уколико је детектована вредност већа од максималне вредности коју приказана величина може имати, како програм реагује на детекцију негативне вредности мерене величине.

За потребе теста омогућена је детекција догађаја са тастатуре. Ово је реализовано имплементацијом функције `on_keyboard()` која је дата листингом 16. Функција као аргументе очекује ASCII код тастера који је притиснут на тастатури и `x` и `y` координату позиције на екрану на којој је детектован клик мишем. За потребе овог теста није коришћен миш, тако да су обрађивани само догађаји код којих је детектован притисак тастера. Да би ови догађаји били прослеђени функцији, неопходно је да она буде регистрована као функција повратног позива за догађаје са тастатуре. То се постиже кодом у листингу 15 који се налази у `main()` функцији.

```
1 glutKeyboardFunc(on_keyboard);
```

Листинг 15: Регистровање функције повратног позива

```
1 static void on_keyboard(unsigned char key, int x, int y)
2 {
3     int i;
4
5     switch (key)
6     {
7         case 27://kraj, dugme esc
8             for(i = 0; i < 10; i++)
9                 unexport_pin(lights[i].gpio);
10            close_log_file();
11            exit(0);
12            break;
13        case '1':
14            lights[1].value = (lights[1].value + 1) % 2;
15            log_data(0, "activated parking brake indicator\n");
16            break;
17        case '2':
18            lights[2].value = (lights[2].value + 1) % 2;
19            log_data( 0, "activated brake lining indicator\n");
20            break;
21        case '3':
22            lights[3].value = (lights[3].value + 1) % 2;
23            log_data( 0, "activated brake fluid indicator\n");
24            break;
```

```

25     case '4':
26         lights[4].value = (lights[4].value + 1) % 2;
27         log_data( 0, "activated check engine indicator\n");
28         break;
29     case '5':
30         lights[5].value = (lights[5].value + 1) % 2;
31         log_data( 0, "activated battery charge indicator\n");
32         break;
33     case '6':
34         lights[6].value = (lights[6].value + 1) % 2;
35         log_data( 0, "activated engine oil indicator\n");
36         break;
37     case '7':
38         lights[7].value = (lights[7].value + 1) % 2;
39         log_data( 0, "activated lights indicator\n");
40         break;
41     case '8':
42         lights[0].value = ( lights[0].value + 1 ) % 2;
43         log_data( 0, "activated fog lighth indicator\n");
44         break;
45     case '9':
46         lights[8].value = ( lights[8].value + 1 ) % 2;
47         log_data( 0, "activated turn left indicator\n");
48         break;
49     case '0':
50         lights[9].value = (lights[9].value + 1) % 2;
51         log_data( 0, "activated turn right indicator\n");
52         break;
53     case '-':
54         if (speed > -1)//testiranje reagovanja na greske
55             speed--;
56         log_data( 0, "activated speed %d\n", speed);
57         break;
58     case '=':
59         if (speed < 250)
60             speed++;
61         log_data( 0, "activated speed %d\n", speed);
62         break;
63     case 'a':
64         if (fuel > -1)
65             fuel--;
66         log_data( 0, "activated fuel %d\n", fuel);
67         break;
68     case 'q':
69         if (fuel < 101)
70             fuel++;
71         log_data( 0, "activated fuel %d\n", fuel);
72         break;

```

```

73     case 's':
74         if (collant_tmp > -1)
75             collant_tmp--;
76         log_data( 0, "activated collant_tmp %d\n", collant_tmp);
77         break;
78     case 'w':
79         if (collant_tmp < 101)
80             collant_tmp++;
81         log_data( 0, "activated collant_tmp %d\n", collant_tmp);
82         break;
83     case 'd':
84         if (tacho_val > -1)
85             tacho_val--;
86         log_data( 0, "activated tacho_val %d\n", tacho_val);
87         break;
88     case 'e':
89         if (tacho_val < 8)
90             tacho_val++;
91         log_data( 0, "activated tacho_val %d\n", tacho_val);
92         break;
93     case 'f':
94         if (odom_val > -1)
95             odom_val--;
96         log_data( 0, "activated odom_val %d\n", odom_val);
97         break;
98     case 'r':
99         if (odom_val < 1000000)
100             odom_val++;
101         log_data( 0, "activated odom_val %d\n", odom_val);
102         break;
103     default:
104         break;
105 }
106
107 glutPostRedisplay();
108 }

```

Листинг 16: Реаговање на догађаје са тастатуре

Главни део `on_keyboard()` функције је `switch` команда која у зависности од притиснутог тастера мења вредности мерених величина или мења вредности за приказ индикатора. На слици 16 приказано је који тастер утиче на коју величину. За вредности које се не односе на индикаторе могуће је задати и вредност за један већу од максималне вредности мерене величине или вредност -1. Овиме се тестира понашање програма у случају грешке која узрокује јављање вредности ван дефинисаног опсега.



Слика 16: Контроле на тастатури за тестирање графичког интерфејса

8.2 Тестирање пинова опште намене

Пинови опште намене прикупљају информације на основу којих се индикатори исцртавају на екрану. Тестирање ових пинова реализовано је везивањем помичног прекидача са пином. Померањем дугмета на прекидачу пропушта се струја до пина и индикатор на који се пин односи се исцртава на екрану. Схемом на слици 10 у поглављу 6 приказано је како су везане компоненте за тест пина који се односи на индикатор дугог светла. На исти начин тестирани су и остали пинови, а овај приступ искоришћен је за симулацију догађаја на прототипу.

8.3 Тестирање комуникације по I²C протоколу

За тест комуникације са ардуином модификован је код `sendData()` функције тако да шаље константну вредност. Модификација је дата у листингу 17. Ова верзија функције коришћена је све док комуникација није успостављена успешно, тако да се на Raspberry Pi-у детектују очекивани подаци. Подаци су модифико-

вани више пута, изостављањем делимитера и мењањем дужине тагова како би се проверила исправност функције за парсирање. Крајња верзија програма све описане тестове пролази успешно.

```
1 void sendData() {  
2   char data[64];  
3  
4   sprintf(data, "TMP52|FUE29|TAC3|SPD43|ODM123456|");  
5  
6   Wire.write(data);  
7 }
```

Листинг 17: Тестна верзија кода на ардуину

9 Закључак

У раду је описан развој прототипа софтвера за обраду и приказ информација у моторном возилу. Почетак рада бави се анализом постојећег система, што је у овом случају аналогна контролна табла. Детаљно је описан процес добијања вредности за сваку контролисану величину. У процесу анализе посебна пажња поклоњена је могућностима хардвера на ком се развијени прототип извршава. Развијени софтвер се првенствено извршава на Raspberry Pi-у и покреће се аутоматски приликом паљења уређаја, одмах након стартовања десктоп окружења. Мањи део софтвера намењен прикупљању података извршава се на Arduino микроконтролеру.

Развијени код написан је у C програмском језику. За графичке потребе рада коришћен је OpenGL. Тестови су подељени у три групе, зависно од тога шта је предмет тестирања. Све три групе дале су позитивне резултате. Утврђено је да развијени софтвер испуњава очекивања и да су испуњени захтеви дефинисани у раној фази развоја.

Данашње контролне табле поред приказа информација о тренутном стању возила имају и прегршт других могућности. Неке од њих су тренутно време, спољашња температура, навигација. У случају да се појави потреба додатног приказа, приказ неке од додатних функционалности могао би бити смештен на централни део екрана, испод брзине. У случају потребе додавања новог индикатора упозорења, могуће је креирати још један ред са индикаторима изнад постојећих.

Описани софтвер, развијен у оквиру рада на мастер тези, доступан је као софтвер отвореног кода. Целокупан код налази се на адреси <https://github.com/TijanaJordanov/CarDashboardPrototype>.

Литература

- [1] 2016 PRODUCTION STATISTICS. <http://www.oica.net/2016-q2-production-statistics/>. Accessed: 2017-06-25.
- [2] Benz Velo and Comfortable, 1894 - 1901. https://mercedes-benz-publicarchive.com/marsClassic/instance/ko.xhtml?grp=INFOTYPE_PKW_BR.DEFAULT_TEXT_GR.&oid=4355#toRelation. Accessed: 2017-06-25.
- [3] Raspberry Pi 2 model b. <https://www.raspberrypi.org/products/raspberry-pi-2-model-b/>. Accessed: 2017-06-14.
- [4] Simple guide to the rpi gpio header and pins. <https://www.raspberrypi-spy.co.uk/wp-content/uploads/2012/06/Raspberry-Pi-GPIO-Layout-Model-B-Plus-rotated-2700x900.png>. Accessed: 2017-09-03.
- [5] What is arduino? <https://www.arduino.cc/en/Guide/Introduction>. Accessed: 2018-05-14.
- [6] Јожеф Декањ. *Електрични уређаји у аутомобилу*. Техничка књига, 1996.
- [7] BMW AG. *Owner's Manual for the vehicle*. BMW AG, 1997.
- [8] Gareth Halfacree and Eben Upton. *Raspberry Pi User Guide*. Wiley Publishing, 1st edition, 2012.
- [9] Konrad Reif. *Gasoline Engine Management*. Springer Vieweg, 2015.

Листинзи

1	Додавање скрипта у листу за покретање у <code>autostart</code> датотеци	24
2	Садржај скрипта <code>dashboard_start.sh</code>	24
3	Функције за рад са дневник-датотекама	27
4	Део кода из функције <code>initialize()</code> за креирање нити	27
5	Функција за брисање дневник-датотека	27
6	Код за читање вредности пина	29
7	Код на Ардуину	30
8	Функција <code>on_timer()</code>	32
9	Код на Raspberry Pi уређају за комуникацију са Ардуином	33
10	Функције за контролу приказа на екрану	34
11	Део функције <code>display</code>	35
12	Део функције <code>draw_center_of_scene()</code>	37
13	Дефиниција структуре <code>Light</code>	38
14	Исцртавање брзине	38
15	Регистровање функције повратног позива	40
16	Реаговање на догађаје са тастатуре	40
17	Тестна верзија кода на ардуину	44