Automatizirano testiranje informacijsko-zabavnog sustava vozila

Matančić, Leo

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU

FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA

Sveučilišni diplomski studij Automobilsko računarstvo i komunikacije

Automatizirano testiranje informacijsko-zabavnog sustava vozila

Diplomski rad

Leo Matančić

Osijek, 2019.

Sadržaj

1. UVOD 1

2. IZAZOVI RAZVOJA I TESTIRANJA PROGRAMSKE PODRŠKE 2

2.1. V Model 2

2.1.1. Testiranje unutar V modela 3

2.1.2. Automatizirano dinamičko testiranje 8

2.2. Standardi informacijsko-zabavnih sustava vozila 9

2.3. Postojeći specijalizirani alati za testiranje 11

3. SPECIFIČNOSTI RAZVOJNE OKOLINE 15

3.1. Sklopovlje razvojnog računala 15

3.2. Programsko okruženje operacijskog sustava 16

3.2.1. Razvojni OS i pripadajući pogonski programi 16

3.3. Razvojna i testna okolina i alati 16

3.3.1. Qt Creator 16

3.3.2. QtTest za testiranje klasa 21

3.3.3. QtQuickTest za grafička sučelja 22

3.3.4. Squish 22

3.3.5. ZeroMQ 23

4. RAZVOJ APLIKACIJE I TESTIRANJE 25

4.1. Razvoj aplikacije za simulaciju ulaza informacijsko-zabavnog sustava vozila 25

4.2. Implementacija ZMQ sučelja unutar vozila 34

4.3. Implementacija testnog alata 36

4.3.1. Testiranje korištenjem Qt testiranja jedinica 36

4.3.2. Testiranje korištenjem alata Squish 37

4.4. Pisanje testnih slučajeva 39

4.4.1. Testiranje jedinica 39

4.4.2. Integracijsko testiranje 43

4.4.3. Sustavsko testiranje 45
4.5. Sučelje aplikacija koje se testiraju 46
5. REZULTATI TESTIRANJA S ANALIZOM 47
5.1. Prikaz pokrivenih testnih slučajeva 47
5.2. Rezultati testiranja jedinica 48
5.3. Rezultati integracijskog testiranja 54
5.4. Rezultati testiranja sustava 68
6. ZAKLJUČAK 70
7. LITERATURA 71
SAŽETAK 74
ABSTRACT 75
ŽIVOTOPIS 76

1. UVOD

Prvi prethodnici informacijsko-zabavnih sustava vozila [1] smatraju se prvi radio sustavi unutar vozila četrdesetih godina prošlog stoljeća, dok se stvarni začetci ovakvih sustava smatra vrijeme oko 2000. godine kada se pojavljuju prvi zasloni na dodir koji omogućuju puštanje zvukovnog sadržaja i GPS navigaciju. Zbog potreba korisnika, ti su sustavi do danas uvelike napredovali, daju korisniku mogućnost upravljanja nad cijelim vozilom i prikaz izvršenih naredbi, davanje povratnih informacija korisniku o stanju sustava unutar i izvan vozila koristeći različite senzore i kamere, pružaju zabavne sadržaje vozaču te putnicima u vozilu kao što su televizija, pretraživanje interneta, slušanje glazbe i sl. kroz različite zaslone, zvučnike i slušalice, daju detaljne informacije o položaju vozila. Samim tim raspolažu puno većim memorijskim kapacitetom, komuniciraju s drugim sustavima unutar vozila, povezani su na Internet pa čak i međusobno sa drugim vozilima, a omogućuju i nadogradnju sustava vozila te brže dostavljanje sigurnosnih zakrpa. Da bi se potvrdilo da jedan takav sustav radi ispravno, odnosno da je siguran za ugradnju u vozilo i da će osigurati najbolje korisničko iskustvo vožnje, takav sustav mora biti testiran.

Cilj ovog diplomskog rada je osmisliti i provesti automatizirano testiranje grafičkog korisničkog sučelja (engl. *GUI, Graphical User Interface*) centralnog zaslona i digitalne nadzorne ploče (engl. *cluster display*). U tu svrhu kreirana je aplikacija koja se pokreće na razvojnom računalu i uz alate za testiranje izvršava automatizaciju testiranja grafičkog korisničkog sučelja na centralnom zaslonu i digitalnoj nadzornoj ploči vozača informacijsko-zabavnog sustava vozila Rimac Concept_One. Aplikacija sadrži simulaciju mehaničkih ulaza te ulaza sa zaslona na dodir koji se testira te i koji će omogućiti snimanje i ponavljanje provedenih testova. Nakon provedenih ulaza u testni alat i izvršavanja testova, koristeći navedene zaslone i aplikaciju za simulaciju navedenih ulaza, iz alata za testiranje korisnik dobiva povratnu informaciju o padu ili uspješnom prolasku testova unutar sučelja alata.

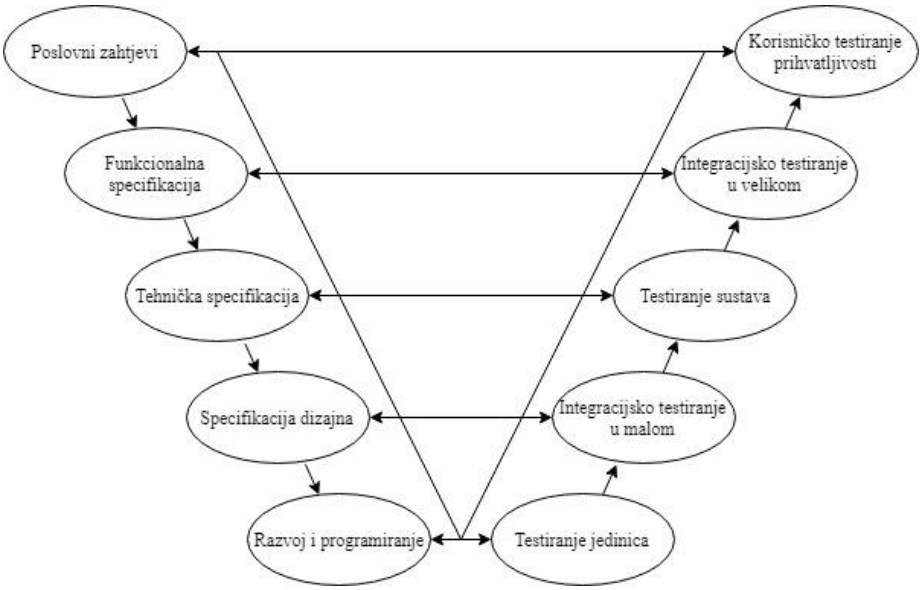
U poglavlju 2 opisani su izazovi životnog ciklusa programske podrške ovakvog rada. Teorijska osnova u poglavlju 3 opisuje detaljno korišteno sklopovlje i programsku podršku koja je korištena za izradu aplikacije, komunikaciju među aplikacijama te njihovo testiranje. Poglavlje 4 opisuje izradu aplikacije za simuliranje mehaničkih ulaza informacijsko-zabavnog sustava vozila, njezinu komunikaciju preko ZMQ komunikacije sa aplikacijama zaslona unutar vozila te sadrži primjere provedenih testova. U poglavlju 5 prikazani su izvedeni testovi te su dobiveni rezultati analizirani.

2. IZAZOVI RAZVOJA I TESTIRANJA PROGRAMSKE PODRŠKE

U ovom poglavlju bit će opisani glavni pojmovi diplomskog rada kao što su životni ciklus programske podrške unutar V modela, automatizirano dinamičko testiranje sustava grafičkog korisničkog sučelja koje će se provesti te standardi koje ovakav sustav informacijsko-zabavnog sustava mora zadovoljiti. Također, bit će ukratko opisana postojeća rješenja specijaliziranih alata za ovakvo testiranje.

2.1. V Model

V model [2] predstavlja grafičku reprezentaciju rada na životnom ciklusu procesa, u ovom slučaju razvoj automobila, preciznije u diplomskom radu testiranje rada grafičkog sučelja informacijsko-zabavnog sustava. Dobio je naziv po izgledu grafičkog prikaza koji poprima oblik slova V, pri čemu se na lijevoj strani definiraju ulazi projekta, u samom špicu se odrađuje implementacija definiranih tehničkih i korisničkih zahtjeva te na desnoj strani se vrši testiranje i integracija pojedinih definiranih specifikacija. Model je dosad široko prihvaćen zbog svoje jednostavnosti, redoslijeda operacija prema slici 2.1 [3] jer omogućavaju definiranje aktivnosti prije samog razvoja i programiranja čime se štedi vrijeme na popravljanju i pronalasku nedostataka te je time model uspješniji od svog prethodnika, vodopadnog modela. Samim korištenjem ovakvog načina rada sprječava se nagomilavanje grešaka ako su zahtjevi razumljivi i dobro precizirani. Međutim ovakav model ne omogućava razvoj ranih inačica programske podrške već ona tek nastaje u fazi implementacije, koja se u slučaju promjena dokumentacija testova i zahtjeva mora nadopuniti, što je jedan od nedostataka zbog kojih je model slabo fleksibilan.



Slika 2.1. *Grafički prikaz V modela*

2.1.1. Testiranje unutar V modela

Modeli testiranja [4] koji se izvode su:

- Verifikacija – ispunjava li programska podrška zahtjeve i uvjete prethodne faze odnosno daje odgovor na pitanje: „Jesmo li ispravno izgradili sustav?“
- Validacija – njome se utvrđuje jesu li zahtjevi u skladu sa krajnjim izgrađenim sustavom ili programskom podrškom odnosno daje odgovor na pitanje: „Jesmo li izgradili pravi sustav?“

Postoje četiri razine testiranja unutar V modela[5]:

- Testiranje jedinica
- Integracijsko testiranje (u malom i velikom)
- Testiranje sustava
- Korisničko testiranje prihvatljivosti

Glavni cilj praktičnog dijela bit će obaviti testiranje jedinica, funkcionalne integracije u malom i sustava grafičkog korisničkog sučelja centralnog i *cluster* zaslona automobila Rimac Concept_One.

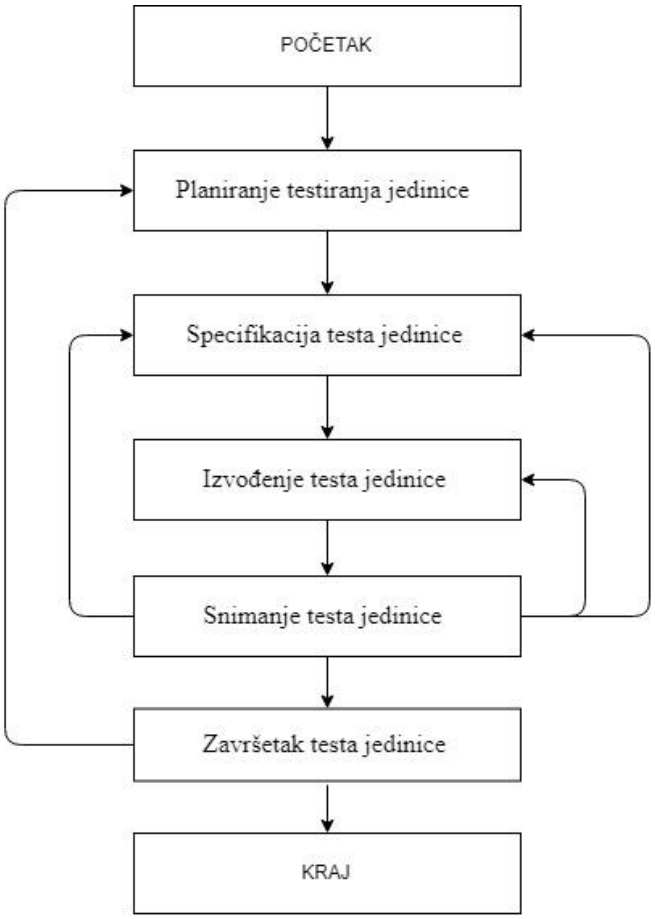
Testiranje jedinica [3] obuhvaća testiranje minimalne jedinice koja može biti testirana u izolaciji. Time se postiže da jedinica funkcionira odgovarajući njezinoj specifikaciji. Najčešće se odvija čim se kod napiše, a najčešće ju obavlja razvojni programer koji poznaje programski kod te upravo on najbolja osoba za pronaći i ispraviti sve pronađene pogreške. Pošto testiranje jedinica obuhvaća samo testiranje jedne po jedne komponente, osoba koja ju testira mora koristiti vrstu omotača (engl. *wrapper*) koji omogućuje komponenti da radi samostalno. Zbog toga se koriste test snopovi (*engl. test harness*) i upravljači testova (engl. test driver). Omotač se obično piše specifično za komponentu ili se koristi komercijalni alat, a služi za unos podataka u jedinicu, vrati podatke iz jedinice te ih prezentira osobi koja testira jedinicu.

Jedan od postojećih pristupa za automatizaciju ovakvog testa je podatcima predvođeno testiranje (engl. *data-driven test*) koji se temelji na stvaranju tablice podataka koja sadrži vrijednosti unosa u jedinicu te očekivane izlazne vrijednosti, pri čemu se kod izvođenja testa te vrijednosti čitaju iz same tablice podataka.

Britanski standard za testiranje jedinica je BS7925-2, a definira i daje detalje o tehnikama testiranja i mjerilima koja se mogu koristiti pri testiranju jedinica unutar okvira metode testiranja crnom kutijom (engl. *black box*) i bijelom kutijom (engl. *white box*).

Proces testiranja jedinica prema slici 2.2 [3] sastoji se od:

1. Planiranja testiranja jedinice – plan napisan s obzirom na specifikaciju programa ili jedinice, najčešće ga piše sam razvojni programer ili analitičar pri čemu opisuje postupak opsega testiranja jedinice, koji testovi se provode i koje uvjete okoline se mora zadovoljiti
2. Specifikacija testa jedinice – ovaj dio testiranja odnosi se točke donošenja odluka i puteva podataka kako se procesiraju, odnosno kako je izvedena pokrivenost koda u testiranju jedinice, a radi se zbog smanjenja vremena izvođenja testiranja jedinica te uklanjanja nedostataka prije nego se jedinica izvrši
3. Izvođenje i snimanje testa jedinice – postupak u kojem se provjerava odgovara li vrijednost izlaza iz testa jedinice očekivanoj vrijednosti, rezultat je prolazak ili pad testa, a pošto je postupak izvođenja često iterativan radi se izrada ponovljivog testa (snimke)
4. Završetak testa jedinice – u ovom koraku se procjenjuje završetak testiranja jedinice odnosno da li je odrađeno izvršenje svih testova, pregled svih rezultata i pregled da su testovi ispravno odrađeni ili je vrijeme bilo ograničeno te su pokrenuti samo prioritetni testovi i odrađeni ispravno



Slika 2.2. *Proces testiranja jedinice*

Integracijsko testiranje [3] radi se kako bi se pronašli i prikazali nedostatci u sučelju između interakcije integriranih komponenti ili sustava. Pri izvođenju ovih testova mora se u obzir uzeti dizajn programske podrške i sustava, arhitektura, tijekovi rada i slučajevi korištenja.

Tipični objekti testiranja u integracijskom testiranju su :

- Podsustavi implementacije baze podataka
- Infrastruktura
- Sučelja

Postoje dvije razine integracijskog testiranja a to su:

1. Integracijsko testiranje u malom – odnosi se na testiranje sučelja i interakcije između integriranih komponenata, a cilj im je dokazati da su podatci preneseni između komponenata kako je zahtijevano te kako bi se pokazalo koliko dobro skup jedinica radi zajedno, pri čemu se za komponente koje su potrebne, ali nisu uključene u test koriste test snopovi i test oslonci (engl. *test stub*) koji repliciraju nedostajuće jedinice.
2. Integracijsko testiranje u velikom odnosi se na testiranje sučelja prema vanjskim organizacijama kao što su elektronička razmjena podataka i Internet te se provodi nakon testiranja sustava, cilj im je utvrditi da su podatci ispravno preneseni između aplikacija i vanjskog sučelja

Planiranje integracijskog testiranja razmatra voditelj testiranja te razmatra u nekoliko točaka životnog ciklusa kako bi zaključio koliko je ono pogodno i što sve treba uključiti. Pri tome se mora donesti odluka kada i na kojim razinama će se integracijsko testiranje provoditi, postoje li neke prepreke, zatim svi elementi testiranja trebaju slijediti sličan postupak kako bi testiranje integracije podržalo poslovni proces.

Postoji više pristupa integracijskom testiranju, a to su:

1. S povećanjem opsega

- Odozgo prema dolje – testiranje se izvodi od komponente na vrhu hijerarhije prema manjima koje su često zamijenjene sa osloncima
- Odozdo prema dolje – testiranje se provodi s najniže razine te se odvija sve dok se ne testira najviša komponenta hijerarhije
- Funkcionalno – odrađuje se analiza funkcionalnosti integracije, kompromis je između testiranja odozgo prema dolje i odozdo prema gore te je najčešće korištena metoda

2. Bez povećanja opsega

- „Veliki prasak“ (*engl. Big Bang*) – sve komponente se integriraju zajedno i sve se testira kao cjelina

BS7925-1 definicija testiranja [3] sustava glasi „Proces testiranja integriranog sustava za verifikaciju da on zadovoljava specificirane zahtjeve“.

Osnove testa sustava su:

- Specifikacija zahtjeva sustava i programske podrške
- Slučajevi uporabe (engl. *use cases*)
- Funkcionalna specifikacija
- Izvještaj analize rizika

Tipični objekti testiranja sustava su:

- Priručnici sustava, korisnika i rada
- Konfiguracija sustava

Testiranje sustava [3] najčešće se odvija nakon integracijskog testiranja u malom, pri čijem se testiranju koriste moduli ili podsustavi koji su do tad napravljeni, dok u testiranju sustava cijeli sustav je izgrađen te se odvija test na njemu ili nekoj njegovoj komponenti. Pri testiranju sustava provjeravaju se funkcionalni i nefunkcionalni zahtjevi rada koji su navedeni u specifikacijama, mada to i ne mora biti slučaj jer nekada nisu sve specifikacije navedene ili nisu točno definirane, što može dodatno otežati posao testiranja. Najčešći i najprihvatljiviji pristup testiranja funkcionalnih specifikacija sustava je pristup crne kutije kojom se može izraditi tablica odluka kombinacije učinaka iz pravila poslovanja, a tek se onda tehnika temeljena na uvidu u strukturu kao što je bijela kutija koristi za procjenu temeljitosti ispitivanja na strukturne elemente, kao što su grafički izbornici i prikazi unutar vozila.

Samo testiranje dijeli se na:

1. Funkcionalno testiranje čija BS7925-1 definicija glasi „Dokument koji opisuje u detalje karakteristike proizvoda s obzirom na njegovu namijenjenu sposobnost.“
   Postoje dvije metode pristupa ovom testiranju:

   - Testiranje utemeljeno na funkcionalnim zahtjevima – izvedeno je iz specifikacije zahtjeva prije razvojne faze i koristi se kao izvor informacija za kreiranje test aplikacije

pri čemu test aplikacija utvrđuje zadovoljava li testirana aplikacija navedene zahtjeve ili ne. Bitno je još jednom provjeriti i utvrditi s korisnikom slaže li se specifikacijom zahtjeva s planom testiranja i testnim slučajevima.

- Testiranje utemeljeno na poslovnom procesu – temelji se na korisnički definiranim scenarijima oponašajući poslovni proces u testovima. Testni slučajevi i njihov razvoj rade se kroz intervju sa korisnikom čime se postiže da ovakav oblik testiranja odražava stvarnu ili barem očekivanu primjenu aplikacije.

2. Nefunkcionalno testiranje ili testiranje kvalitete odnosno testiranje značajki kvalitete – odnosi se na testiranje komponenata sustava koje ne utječu na funkcionalnost (kao što su pouzdanost, efikasnost, korisnost, održivost i prenosivost) već dobivanje korisničke pažnje na značajke koje razvijena aplikacija nudi, a sam proces nefunkcionalnog testiranja odvija se zbog potrebe za što boljim korisničkim iskustvom u radu sa aplikacijom, odnosno kako bi se razvijena aplikacija bila spremna za uspješnu implementaciju.
   Neki od ovih testova su:

   - Test korisnosti – rješava pitanje koliko je aplikacija jednostavna za snalaženje kroz izbornike te intuitivna za korištenje, a obično se testira s novim korisnicima koji prvi puta vide aplikaciju
   - Testiranje pohrane – testiranje aplikacije na potrebe korištenja prostora za pohranu te može li se taj prostor efektivnije koristiti.
   - Ispitivanje mogućnosti instalacije – testiranje jednostavnosti instalacije aplikacije, rada aplikacije nakon instalacije, rada aplikacije na različitim konfiguracijama te testiranje ispravnog brisanja aplikacije
   - Testiranje dokumentacije – testiranje napisane dokumentacije na neispravnosti ili neprecizna može navesti korisnike prijavu nedostataka u proizvodu kada on ne radi neispravno
   - Testiranje oporavka – testiranje sustava na namjerno rušenje kako bi pristupio svojim mogućnostima oporavka te testiranje mjera koje se poduzimaju, kako se dođe do oporavka iz perspektive korisnika te koliko je vremena i truda potrebno da se sustav vrati u rad sa svim svojim podatcima. Najčešće se testira na kratki pad mreže tj. isključivanje sustava gubitkom električne energije.
   - Testiranje opterećenja – testiranje aplikacije na rukovanje sa svakodnevnim zadatcima sa svim korisnicima na temelju poslovne uporabe se mjere odzivi aplikacije

- Testiranje na otpornost od stresnog opterećenja – provodi se testiranje većih cjelina sustava ili aplikacija na krajnjoj granici mogućnosti kako bi se procijenilo trajnost pod stresom, odnosno računa se vrijeme prije pada (engl. *MTTF – Mean Time to Failure*)
- Testiranje performansi – testiranje na performanse kao što su odziv sustava u radu itd.
- Testiranje na obujam podataka – testira kako se sustav ponaša na rukovanje s velikim količinama podataka

2.1.2. Automatizirano dinamičko testiranje

Dinamičko testiranje [6] postupak je testiranja koda u samom njegovom izvođenju, odnosno kada se program nalazi u izvršnom okruženju (engl. *runtime environment*). Programski kod se testira na određene ulaze i uspoređuju se dobivene vrijednosti s očekivanim izlazima čime se postiže validacija. Pri tome mogu se mogu promatrati:

- funkcionalni parametri programske podrške
- nadziranje memorije sustava
- odziv centralne procesorske jedinice i
- performanse sustava

Automatizacija testiranja [7] je postupak korištenja programske podrške koja je odvojena od programske podrške koja se testira kako bi se kontroliralo izvođenje testova i dobilo usporedbu dobivenih i očekivanih rezultata, pri čemu testovi su vrlo često repetitivni, ali važni. Zbog toga je od velike važnosti automatizacija testova jer se njome postiže pristup razvoja programske podrške kroz kontinuiranu integraciju i isporuku (engl. *CI-CD, Continuous Integration – Continuous Delivery*) uz testiranje.

Opći pristupi automatiziranom testiranju su:

- Testiranje grafičkog korisničkog sučelja – korištenje testnog okvira koji generira događaje sa korisničkog sučelja kao što su pisanje na tipkovnici ili klikovi miša te promatrajući te promjene na korisničkom sučelju kako bi se utvrdilo da je ponašanje u skladu s očekivanim, a u nekim slučajevima postoji skriptna varijacija automatizacije testiranja koja pruža mogućnost njihovog snimanja i ponovnog izvođenja
- API vođeno testiranje – korištenje programskog sučelja prema aplikaciji koja se testira kako bi se utvrdila valjanost ponašanja koje se testira, a obično zaobilazi korisničko sučelje te testira sučelja klasa, module ili biblioteke s različitim ulaznim argumentima kako bi se potvrdila točnost vraćenih rezultata.

2.2. Standardi informacijsko-zabavnih sustava vozila

Kako bi se zadovoljila homologacija vozila, vozila moraju zadovoljiti određene standarde automobilske industrije. Iz tog razloga provode se detaljna testiranja sklopovlja i programske podrške svih komponenata vozila, od kojih je jedna i informacijsko-zabavni sustav. Glavna norma po kojoj se izrađuje sklopovlje je LV124 norma koja obuhvaća dvije vrste testova:

- električne zahtjeve i testove 12V sustava u vozilu [8]
- zahtjeve na uvjete u okolini [9]

Jedna od glavnih stavki iz te norme je da zasloni informacijsko-zabavnog sustava koji se koriste u automobilu da bi prošli homologaciju moraju biti automobilskog razreda, pri čemu zadovoljavaju radne temperature od -45°C do 85°C.

Tablica 2.1 *Tablica izračuna ASIL razina*

		Učestalost (engl. Exposure)	Upravljivost (engl. Controllability)		
			C1	C2	C3
Ozbiljnost (engl. Severity)	S1	E1	QM	QM	QM
		E2	QM	QM	QM
		E3	QM	QM	A
		E4	QM	A	B
	S2	E1	QM	QM	QM
		E2	QM	QM	A
		E3	QM	A	B
		E4	A	B	C
	S3	E1	QM	QM	A
		E2	QM	A	B
		E3	A	B	C
		E4	B	C	D

Idući standard koji je potrebno je zadovoljiti je zahtjev za funkcionalnom sigurnosti ISO26262 [10]. Njome se nalaže za određene uvjete koju razinu integriteta automobilske sigurnosti komponenta mora postići (engl. *ASIL, Automotive Safety Integrity Level*). Ta razina dobiva se izračunom rizika iz umnoška ozbiljnosti nezgode (engl. *Severity*) sa razinama od 1 do 3, učestalosti (engl. *Exposure*) kojom se može dogoditi sa razinama od 1 do 4 te upravljivosti (engl.

Controllability) sa razinama od 1 do 3 ako dođe do određenog događaja prema tablici 2.1. Sa upravljanjem kvalitete (engl. *QM, Quality Management*) se označava razina koja se ne ubraja u opasne događaje stoga ne zahtjeva mjere sigurnosti po standardu ISO26262. Zahtjev ASIL razine u našem slučaju za centralni zaslon je QM, dok je zaslon digitalne nadzorne ploče ASIL razine B.

S obzirom da je uređaj namijenjen globalnom tržištu također mora proći testove na elektromagnetske smetnje [11] (engl. *EMI, Electromagnetic Interference*) koje smetaju funkcioniranju elektroničkog uređaja, a kojima su često uzroci uvjeti u okolini, ali i izvori drugih elektroničkih uređaja i sustava. Također moraju se zadovoljiti zahtjevi za elektromagnetskom kompatibilnosti (engl. *EMC, Electromagnetic Compatibility*) tako da uređaj radi kao što je predviđeno u danoj radnoj okolini, a pri tome ne utječe na rad drugih uređaja niti sustava. Oba elektromagnetska testiranja provode se za razna tržišta i dobivanje raznih certifikata, kao što su certifikat potrošačke elektronike (engl. *CE, Consumer Electronics*), Ujedinjenih naroda ekonomske komisije za Europu (engl. *UN/ECE United Nations Economic Commision for Europe*), Federalne komisije za komunikacije (engl. *FCC, Federal Communications Commision*) te kineske obavezne certifikacije (engl. *CCC, China Compulsory Certification*).

Također, kao jedan od bitnih standarda zaštite nalaže se standard IEC *(International Electrotechnical Commision*) za zaštitu elektroničke opreme u kućištima od prodora krutih objekata i tekućina [12] prema tablici 2.2 koja sadrži IP (engl. *Ingress Protection*) vrijednosti. Sustav obilježavanja postignute razine zaštite označava se oznakom IP te dvije znamenke, od kojih je prva zaštita od krutih objekata, a druga zaštita od tekućina.

Tablica 2.2 *IP sustav označavanja*

IP	Prva znamenka: Zaštita od prodora čvrstih predmeta	Druga znamenka: Prodor tekućina
0	Bez zaštite	Bez zaštite
1	Zaštita od čvrstih predmeta preko 50mm	Zaštita od okomito padajućih kapljica vode ili kondenzacije
2	Zaštita od čvrstih predmeta preko 12.5mm	Zaštita od padajućih kapljica vode ako je kućište zakrenuto do 15° od okomice
3	Zaštita od čvrstih predmeta preko 2.5mm	Zaštita od prskanja vode iz bilo kojeg smjera, čak i ako je kućište postavljeno 60° od vertikale
4	Zaštita od čvrstih predmeta preko 1.0mm	Zaštićeno od prskanja vodom iz bilo kojeg smjera

5	Ograničena zaštita od prodora prašine	Zaštita od mlaza vode pod niskim tlakom iz bilo kojeg smjera s dopuštenim ograničenim ulaskom vode
6	Totalna zaštita od ulaska čestica prašine	Zaštita od mlaza vode pod visokim tlakom iz bilo kojeg smjera s dopuštenim ograničenim ulaskom vode
7	Nepoznato	Zaštita od kratkog razdoblja uranjanja u vodu
8	Nepoznato	Zaštita od dugotrajnijih razdoblja uranjanja u vodu
9k	Nepoznato	Zaštita od visokog pritiska te visoko-temperaturnih prskanja

2.3. Postojeći specijalizirani alati za testiranje

Na tržištu već postoje verzije automatiziranih alata za testiranje grafičkih korisničkih sučelja napisanih u QML-u koje će biti opisane u nastavku. Od opisanih primijenjena je komercijalna inačica Squisha za testiranje QML grafičkog korisničkog sučelja nad više aplikacija odjednom u sklopu integracijskog testiranja i testiranja sustava dok se testovi jedinica aplikacija vozila Concept_One provode kroz QtTest razvojni okvir (engl. *framework*) na ulaze iz prethodno kreirane aplikacije koja simulira mehaničke ulaze zabavno-informacijskog sustava samoga vozila.

Ranorex Studio [13] (Slika 2.2 [14]) jedan je od vodećih alata za provođenje automatiziranih testova grafičkih korisničkih sučelja. Tvrtka koja ga razvija je Ranorex GmBH, a razvili su *framework* koji podržava mnoštvo tehnologija za testiranje desktop, mrežno baziranih i mobilnih aplikacija. Trenutna stabilna inačica alata je 9.1.0.

Glavne značajke [15] alata su:

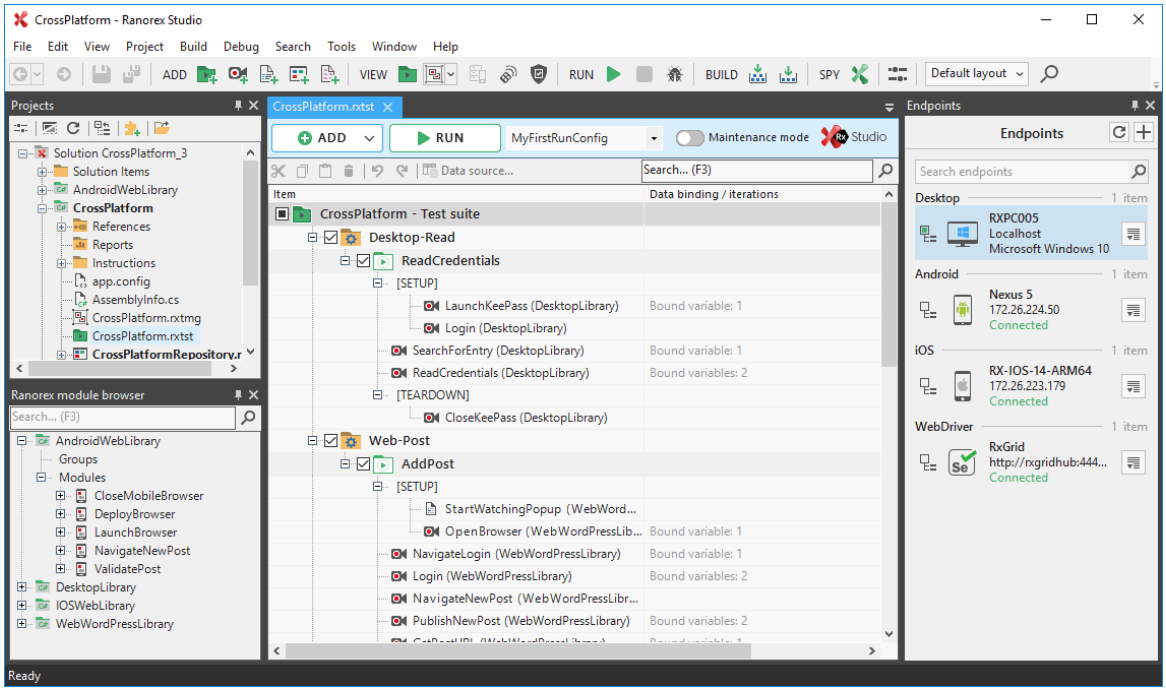
- unakrsno testiranje većine popularnih pretraživača
- istovremeno unakrsno testiranje na pravim uređajima ili simulatorima/emulatorima
- integracija sa CI poslužiteljima, alatima za praćenje zadataka itd.
- XML bazirani izvještaji i analiza te podrška za alate kao JUnit kompatibilne izvještaje o testovima
- robusno rješenje za identifikaciju UI elemenata pomoću značajke Ranorex Spy koja koristi prepoznavanje elemenata RanoreXPath čime se grade pouzdani testovi čak i nakon promjene pozicija elemenata, a omogućuje i prepoznavanje atributa, povezanih elemenata na temelju veza između prethodnih i sljedećih stanja i kreiranje snimaka datoteka

- Ranorex Recorder omogućuje snimanje i ponovno izvođenje testova bez potrebe za programiranjem, pri čemu se mogu i urediti odrađene radnje, dodavati tekst, mijenjati parametri te graditi podatkovno vođeni testovi ili testovi vođeni na ključnim riječima
- Podrška za uređivanje koda u jezicima C# i VB.NET za stvaranje prilagođenih testova potpuno u korisničkom kodu

Alat kao ovakav podržava i automatizaciju testiranja Qt aplikacija [16], pri čemu podržava Qt verzije:

- Qt Widgets od verzije Qt 4.5.3,
- Qt Quick od verzije Qt 5.0.2 te
- QtWebKit od verzije Qt 4.6.4

Neka od ograničenja alata su da podrška vrijedi samo za dinamički povezane Qt aplikacije, ne podržava testiranje mobilnih Qt aplikacija te podrška za desktop testiranje vrijedi jedino za OS Windows te ne podržava unakrsno testiranje na Linux uređaje. Također ako se želi koristiti Qt pristup UI elementima umjesto Ranorex načina pristupa Qt UI elementima mora se uključiti MSAA (engl. *Microsoft Active Accessibility*) u Qt aplikaciji te postaviti Ranorex alat u Qt legacy način automatizacije.



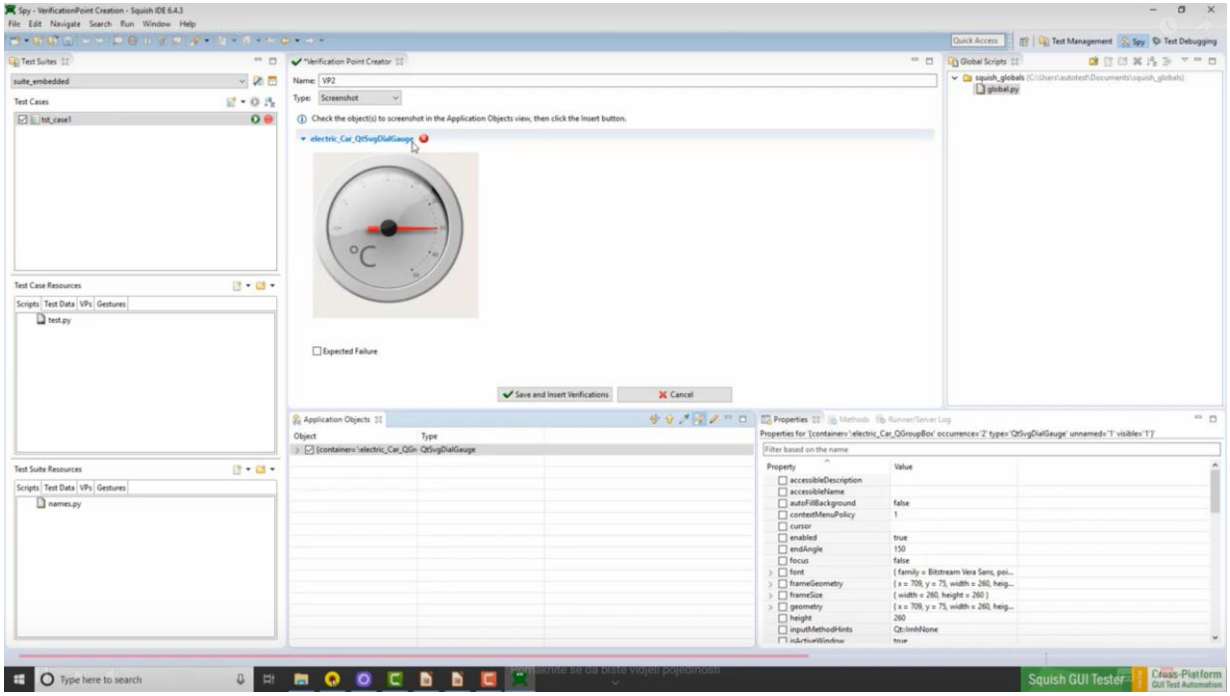
Slika 2.2. *Sučelje alata Ranorex Studio*

Squish [17] (Slika 2.3 [18]) komercijalni je alat tvrtke Froglogic usmjeren na automatizaciju testiranja grafičkih korisničkih sučelja i regresijskih testiranja između različitih platformi. Trenutna stabilna inačica alata je 6.2, a procjena je da ga koristi preko 3000 tvrtki. Razlog tome su njegove značajke [19]:

- Testiranje i razvoj temeljen na ponašanju, BDD (engl. *Behaviour Driven Development & Testing*) koji koristi Gherkin jezik za stvaranje, snimanje, održavanje i uklanjanje neispravnosti
- Snimanje i ponovno puštanje (engl. *Recording and Playback*) koji omogućuju automatsko snimanje i prepoznavanje visoke razine interakcije i objekata umjesto niske razine događaja
- Potvrda testa i provjera valjanosti (engl. *Test Verification & Validation*) koji se temelji na postavljanju točaka provjera, čime se mogu provjeriti vrijednosti objekata, napraviti usporedbe slika ili potvrditi vrijednosti iz tablica
- Jako i intuitivno okruženje za stvaranje testova integrira snimanje, izvođenje, rezultate, otklanjanje neispravnosti skripta, nadziranje objekata te napredno uređivanje i održavanje skripti
- Podrška za velik broj skriptnih jezika kao što su Python, Perl, JavaScript, Tcl i Ruby [20]
- Testiranje predvođeno podatcima (engl. *Data Driven Testing*) iz različitih izvora
- Distribuirano serijsko testiranje (engl. *Distributed Batch Testing*) kako bi se izveo skup skripti ili serija te dobio uvid u detaljne zapise i rezultate izvršavanja
- Opsežne mogućnosti integracije
- Karte objekata i alat za identificiranje objekata
- Hibridno testiranje aplikacija omogućuje jednostavno automatiziranje aplikacija koje koriste više tehnologija ili koje koriste više od jednog alata
- Vizualna verifikacija
- Testiranje zasnovano na slikama koje mogu biti 2D ili 3D grafički prikazi
- Optičko prepoznavanje znakova (engl. *OCR, Optical Charactrer Recognition*) za raspoznavanje teksta s zaslona

Squish podržava Qt testove za sučelja između čovjeka i uređaja [21] (engl. *HMI, Human Machine Interface*) među koja spadaju i grafička korisnička sučelja kako bi se postigao sklad sa sigurnosnim standardima industrije. Koristi ga za testiranja u automobilskoj industriji, medicinskoj avionskoj elektronici, prijevozu, industriji, potrošačkim uređajima, kućanskim aparatima i kućnoj automatizaciji. Razlog tome je potpora za sve Qt Widgets, QML i Qt Quick verzije i kontrole kao

i Qt Webkit te Qt WebEngine, a za HMI testiranje na ugradbenim uređajima posebno je potreban njihov SDK za ugradbene uređaje. Posebno je usmjeren na Qt informacijsko-zabavni sustav vozila kojim pruža C++ klase i QML tipove za pristup značajkama vozila kao i jezgru sučelja za programiranje aplikacija, a dozvoljava i interakciju sa svim značajkama vozila koje pruža Qt IVI modul. Nadalje pruža podršku za uvoz i interakciju sa funkcionalnom modelima jedinica (engl. *FMU, Functional Mockup Units*) i pozadinskih komponenata kao dio automatizacije testiranja sustava, analizu pokrivenosti koda u C, C++ te QML kodu, pokrivenost testovima grafičkog korisničkog sučelja te sučelja čovjek i uređaj, vizualne potvrde nastale kombinacijom provjere svojstava elemenata i slika te automatizirano testiranje između više krajnjih uređaja sa samo jednom skriptom.



Slika 2.3. *Sučelje alata Squish*

3. SPECIFIČNOSTI RAZVOJNE OKOLINE

U ovom poglavlju opisane su specifičnosti sklopovlja razvojnog računala i operativni sustav na kojem je razvijana aplikacija te njegove pojedinosti, a opisani su i sami alat, *framework* i API koji su korišteni za izradu konačnog rješenja diplomskog rada.

3.1. Sklopovlje razvojnog računala

Specifikacije razvojnog računala Lenovo Thinkpad T470p [22] na kojem je izrađen diplomski rad prikazane su u Tablici 3.3.

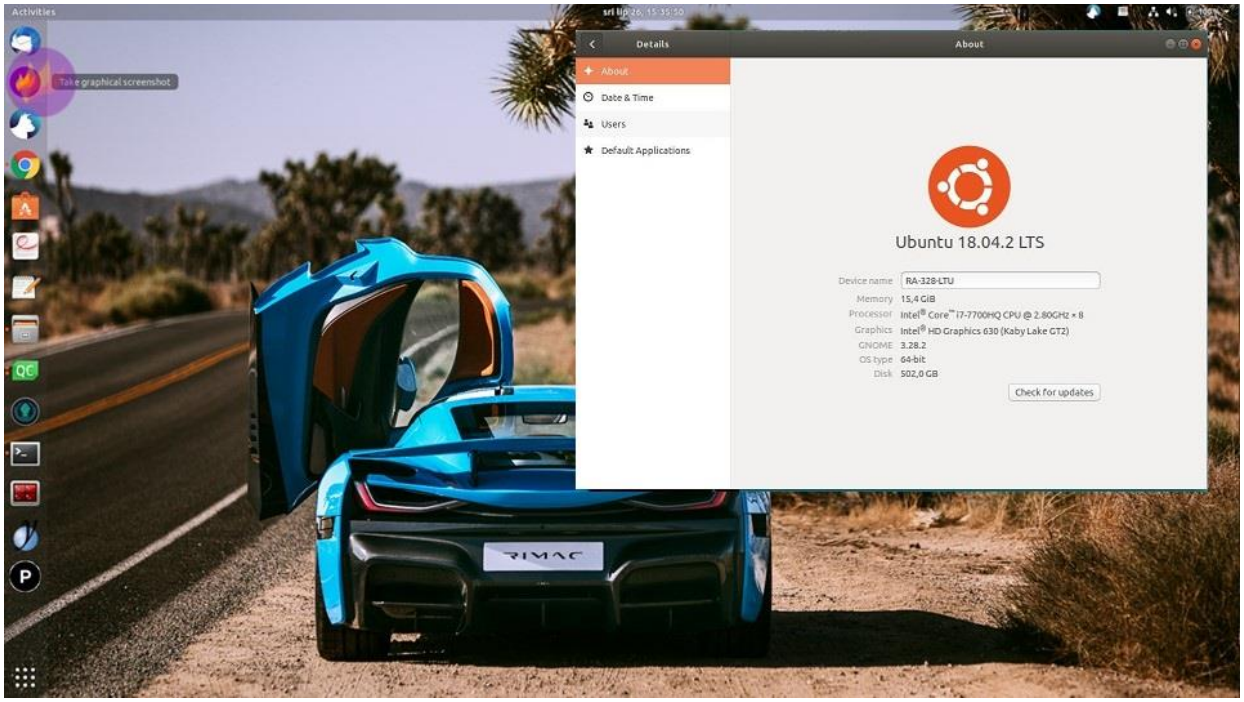
Tablica 3.3 *Tehničke specifikacije razvojnog računala*

Model	Lenovo Thinkpad T470p
BIOS	Lenovo R0FET48W (1.28)
Procesor	Intel® Core™ i7-7700HQ CPU @ 2.80GHz, 1 fizički procesor, 4 jezgre, 8 niti
Radna memorija	Ulaz 0: 1x16GB SODIMM DDR4 2400MHz Ulaz 1: prazan
Pohrana	Samsung SM961 PCIe SSD 512GB
Grafičke kartice	iGPU: Intel HD Graphics 630 3D kontroler: Nvidia GeForce 940MX 2GB, 2.5GHz GDDR5
Zaslon	14.0“ fullHD (1980x1080p), sloj protiv odsjaja, LED osvjetljenje, IPS matrica, 250 nit osvjetljenja, 16:9 omjer, 600:1 kontrast, 160° kut gledanja
Ethernet	Intel Ethernet veza I219-V (Jacksonville)
WLAN	Intel 8265, Wi-Fi 2x2 802.11ac + BT4.1, M.2 kartica
Bluetooth	Intel8265ac 4.1 Wireless integriran u Wi-Fi + Bluetooth adapter
Baterija	SMP 45N1736 Li-ion 47520mWh, 10.8V
Izlazna sučelja	1x Pametni čitač kartica 1x 4-u-1 čitač (MMC,SD,SDHC,SDXC) 3x USB 3.1 1 generacije 1x HDMI 1.4b 1x Mini DisplayPort 1.2A 1x Ethernet (RJ-45) 1x Priključak za priključnu stranicu

3.2. Programsko okruženje operacijskog sustava

3.2.1. Razvojni OS i pripadajući pogonski programi

Na razvojnom računalu prema slici 3.2 pokreče se pomoću GRUB 2.02-ubuntu 8.13 *bootloardera* operativni sustav Ubuntu 18.04 LTS [23] (engl. LTS, *Long Term Support*) odnosno inačica koja ima dugoročnu podršku od 5 godina točnije do travnja 2023.godine. Ovo izdanje [24] Ubuntua pružilo je Linux jezgru (engl. *kernel*) verzije 4.15, sigurnosna poboljšanja te podršku za 32-bit PowerPC.



Slika 3.2. *Ubuntu 18.04 LTS Bionic Beaver*

3.3. Razvojna i testna okolina i alati

3.3.1. Qt Creator

Qt Creator [25] integrirano je razvojno okruženje (engl. *IDE, Integrated Development Environment*) za Qt dostupno za platforme Windows, Mac i Linux, a moguće ga je pokrenuti i na razvojnom računalu. Kroz razvojno okruženje je omogućen razvoj GUI aplikacija, ali i prevođenje programskog koda za desktop verzije, ali i za druge uređaje koristeći unakrsni prevoditelj. Okruženje pruža inteligentno dovršavanje koda, označavanje sintakse, integriran sustav pomoći, alat za uklanjanje grešaka, ugrađeno profiliranje (praćenje memorijskih resursa, kompleksnosti programa, korištenje određenih instrukcija te učestalosti i trajanja funkcijskih poziva kako bi se bolje optimizirao program) te sustave za kontrolu inačica.

Sam Qt [26] je višeplatformni razvojni *framework* napisan u C++, a namijenjen je razvoju desktop, ugradbenih i mobilnih aplikacija. Neke od podržanih platformi su:

- desktop: Linux/X11, macOS, Windows
- mobilne: Android, iOS/tvOS/watchOSq, winRT
- ugradbene platforme: Embedded Linux, INTEGRITY, QNX, VxWorks

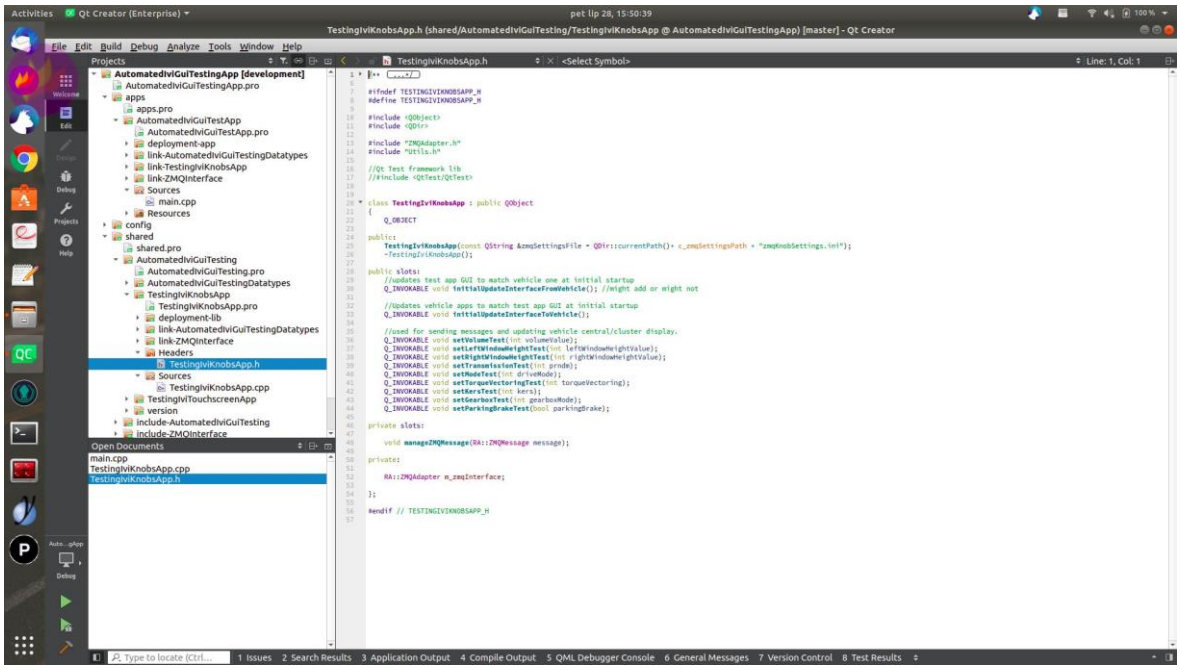
Kao *framework* najvećim dijelom je usmjeren komercijalnom razvoju programske podrške u C++ i QML jeziku za koje pruža velik broj modula [27] prema slici 3.3 [28].

Slika 3.3. *Prikaz Qt modula*

Za razvoj u C++ Qt *frameworku* prema slici 3.4 pruža se velik broj biblioteka i već gotovih aplikacijskih blokova. Neke od dodatnih značajki [29] koje Qt dodaje C++ jeziku su:

- mehanizmi meduobjektne komunikacije zvani signali i utori (engl. *signals and slots*)
- upitnost i svojstva objekata
- događaje i filtere događaja
- kontekstualno prevođenje nizova znakova radi internacionalizacije
- vremenski upravljani tajmeri koji omogućuju integraciju mnogih zadataka u GUIju
- hijerarhijsko i upitljivo objektno stablo koje organizira posjedovanje objekata
- očuvani pokazivači koji su postavljeni na vrijednost 0 čim se referencirani objekt uništi , za razliku od normalnih C++ pokazivača koji postaju viseći pokazivači kada su njegovi objekti uništeni
- dinamičko dodjeljivanje tipova

Zbog tih dodatnih značajki mora se koristiti drugi prevoditelj zvan MOC odnosno *Meta-Object Compiler* koji čita C++ zaglavlja te ako pronađe da klasa sadrži Q_OBJECT makro proizvodi C++ izvorišnu datoteku koja sadrži meta-object kod za te klase. Uz to meta-object kod je potreban za mehanizam signala i utora, informacije o vrsti izvršavanja (engl. *run-time type information*) te dinamička svojstva sustava. Tako generirana C++ izvorišna datoteka pomoću MOC-a mora se prevesti i povezati s implementacijom klase.

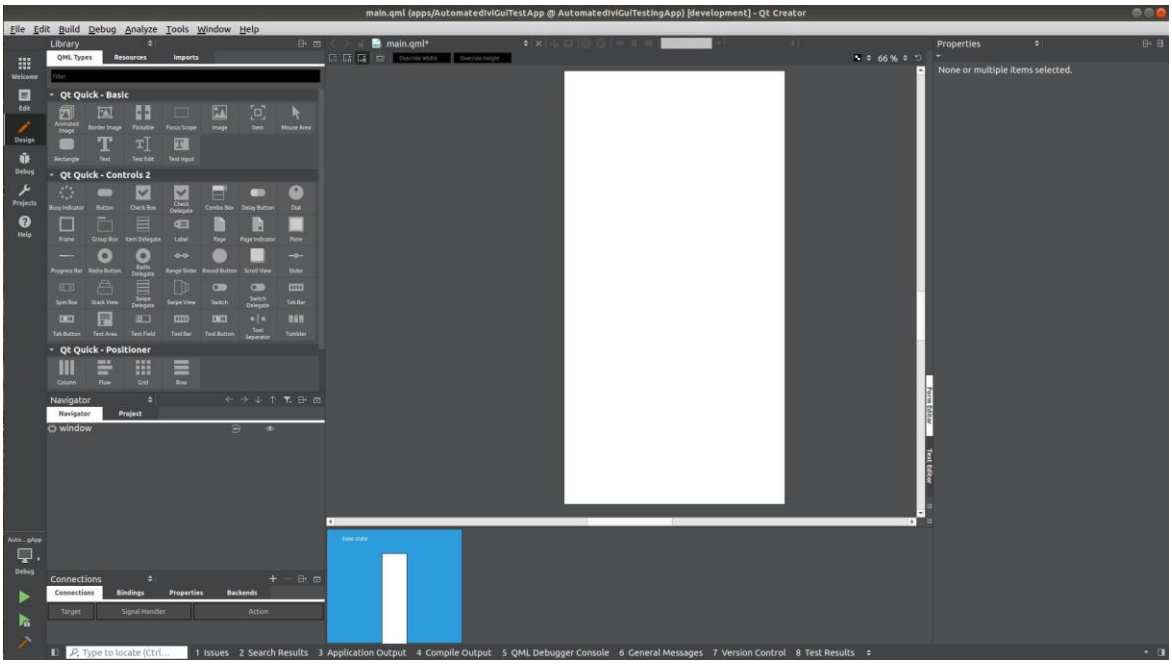


Slika 3.4. *Razvoj u C++ unutar Qt Creatora*

Za razvoj u QML deklarativnom jeziku u Qt Creatoru [30], kojem je podloga Java-Script koristi se Qt Quick standardna biblioteka koja pruža osnovne tipove za kreiranje korisničkog sučelja kroz dizajnerski alat prema slici 3.5 ili kroz pogled za uređivanje dok Qt QML pruža pokretanje QML aplikacija i jezičnu infrastrukturu.

Neke od funkcija Qt Quick biblioteke su:

- pruža QML aplikacijsko programsko sučelje (engl. *API, Application Programming Interface*) koje daje mogućnost korištenja QML tipova čime se pruža definiranje izgleda i ponašanja korisničkog sučelja
- pruža C++ API kao proširenje na QML aplikaciju sa C++ programskim kodom
- omogućava korištenje znanja JavaScript, HTML i CSS za izradu aplikacija
- optimizacije za zaslone osjetljive na dodir te animirana mobilna sučelja

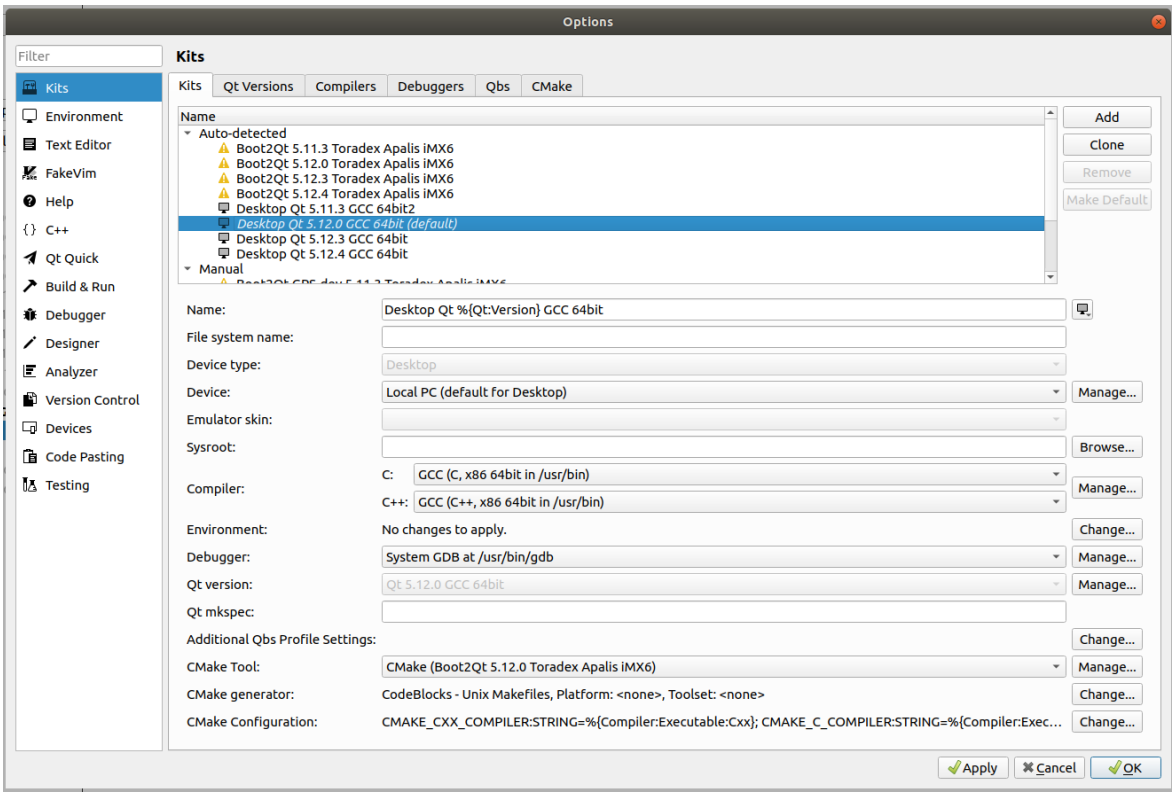


Slika 3.5. *Qt Quick za izradu korisničkog sučelja unutar pogleda za dizajnere*

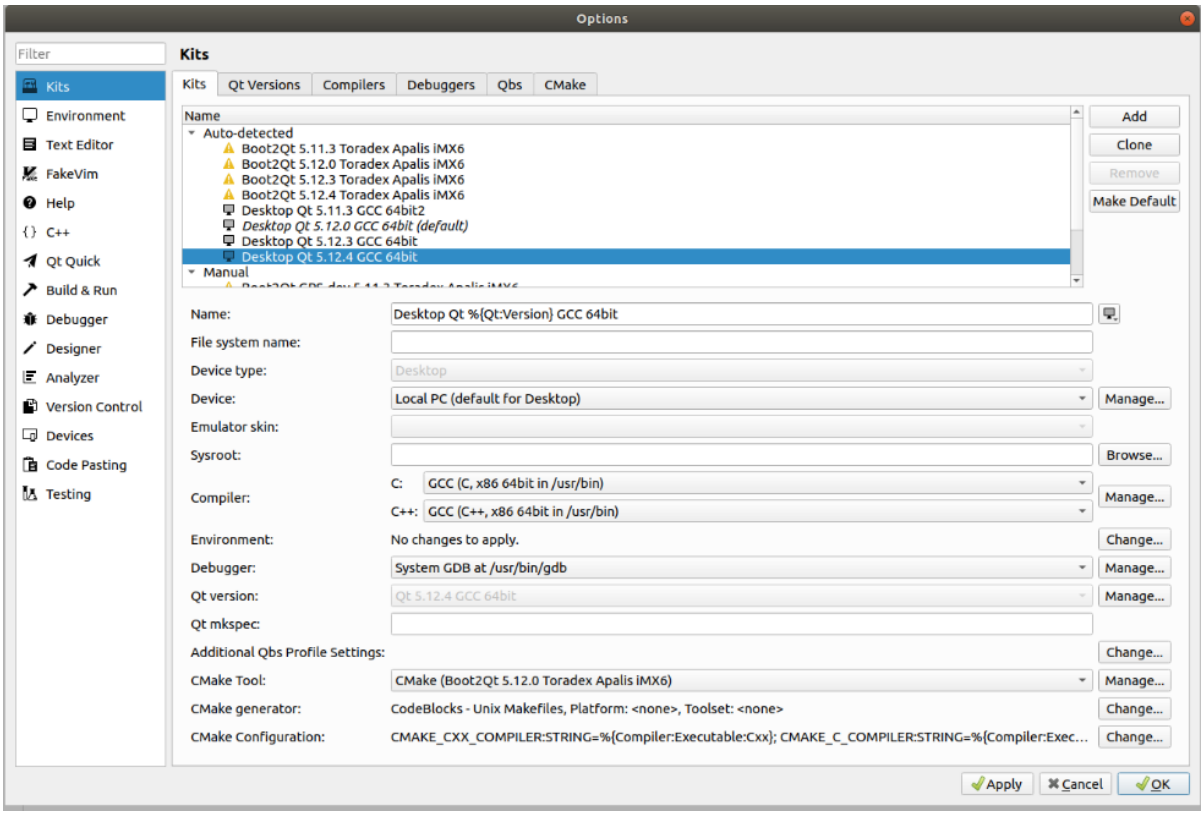
Međutim, Qt i Qt Quick pružaju i mogućnost povezivanja sa drugim jezicima [30] kao što su: Python, Ring, Rust, Crystal, Go, C#/Mono/.Net, D, Haskell, Qtah, Julia, Ocaml, Node.js te Nelson.

Nakon pisanja programskog koda aplikacije on se mora izgraditi za neku od spomenutih platformi. U našem slučaju aplikacija se izgrađuje sa desktop alatom za izgradnju unutar samog Qt Creatora za verzije Qt 5.12.0 korištene za izgradnju aplikacije koja simulira ulaze informacijsko-zabavnog sustava prema slici 3.5 i verzije Qt 5.12.4 korištene za izgradnju aplikacija centralnog zaslona i digitalne nadzorne ploče prema slici 3.6.

Međutim, u slučaju da aplikaciju želimo prebaciti na određeni uređaj, npr. razvojnu ploču s određenim procesorom koji sadrži specifični modul operativnog sustava, tipa Boot2Qt stog programske podrške moramo odabrati točan alat za izgradnju koji smo već dodali u Qt, a to u ovom slučaju treba biti određena inačica Boot2Qt za ugradbeni uređaj.



Slika 3.5. *Qt 5.12.0 postavke alata za izgradnju aplikacije koja upravlja sučeljem*



Slika 3.6. *Qt 5.12.4 postavke alata za izgradnju aplikacija Concept_One*

3.3.2. QtTest za testiranje klasa

QtTest [32] je *framework* za testiranje jedinica aplikacija i biblioteka kojima je baza Qt. Pruža testiranje funkcionalnosti često vidljivih u *frameworkovima* za testiranje jedinica te proširenja za testiranje grafičkih korisničkih sučelja. Kroz sami *framework* pružene su metode koje se nalaze u QTest prostoru imena, koji osim testiranja pruža i uvid u signale i utore kroz QSignalSpy klasu te omogućuje ne-destruktivna testiranja modela premeta kroz QAbstractItemModelTester klasu. Neke od značajki Qt Testa [33] su:

- Lagan - sastoji se od otprilike 6000 linija koda sa 60 izvezenih simbola funkcija
- Samostalan - zahtjeva jako mal broj simbola iz Qt Core modula za testiranja koja nisu vezana za sučelje
- Brzo testiranje - ne zahtjeva posebne nikakve posebne pokretače za testove niti posebne registracije za testove
- Testiranje vođeno podatcima – testiranje se može izvršiti više puta s različitim testnim podatcima
- Osnovno testiranje GUI – nudi funkcionalnost za simulaciju tipkovnice i miša
- Vrednovanje – podržano je vrednovanje i pruža više mjerenja u pozadini
- Prijateljski je za integrirano razvojno okruženje – šalje poruke koje mogu interpretirati Qt Creator, Visual Studio i Kdevelop
- Siguran za niti – izvještavanje o greškama je sigurno i ne utječe na rad niti svojom veličinom
- Siguran za tipove – korištenjem predložaka se sprječava mogućnost grešaka implicitnog postavljanja tipa podataka
- Lako proširiv – prilagođeni tipovi mogu se dodati kao test podatci i test rezultati

Sve što je za potrebe testiranja potrebno je napraviti je dodati u alat za izgradnju odgovarajuću biblioteku te konfiguraciju, dodati biblioteku QtTest u zaglavlje i napisati test slučajeve [34].

Ono što test radi je da pri izvođenju koda aplikacije odradi verifikaciju nad njim što se uobičajeno radi usporedbom dobivene vrijednosti sa očekivanom, što se velikim dijelom radi sa samo dvije makro vrijednosti, a to su:

- QVERIFY makronaredbe i ona ima brojač od 5 sekundi te ako uvjet prođe nastavlja se izvođenje testa, odnosno ako ne prođe ne nastavlja se testiranje
- QCOMPARE makro koristi se kada želimo usporediti dvije vrijednosti te očekujemo da one budu jednake i tada bi se nastavilo izvođenje, dok u suprotnom se izvođenje prekida.

3.3.3. QtQuickTest za grafička sučelja

QtQuickTest [35] je *framework* za testiranje jedinica Qt modelirajućeg jezika (engl. QML, *Qt Modelling Language*) aplikacija. Testovi se pišu koristeći JavaScript funkcije unutar TestCase tipa, a funkcije koje se testiraju započinju sa test_ unutar imena. QtQuickTest nema sigurnost da će se testiranje aplikacije na aplikaciji razvijenoj u jednoj Qt verziji raditi na drugoj verziji. Za potrebe testiranja koristi se istoimena biblioteka QtQuickTest, ali se za samo pokretanje testnih slučajeva koristi C++ snop koji se sastoji od pretprocesorske naredbe za uključivanje spomenute biblioteke te QUICK_TEST_MAIN makro naredbu sa argumentom unutar zagrade koji navodi ime testne klase koju pokreče C++. Međutim ako je potrebno testirati i C++ programski kod koji komunicira sa QML sučeljem tada je potrebno koristiti QUICK_TEST_MAIN_WITH_SETUP makro naredbu kako bi se moglo postaviti svojstvo kontekstnog svojstva iz C++ datoteke prema QML sučelju radi njihovog povezivanja, pri čemu je toj makro naredbi potrebno predati dodatni argument tipa QObject* te se u samoj C++ datoteci treba dodati pretprocesorska naredba za uključivanje prevedene .moc datoteke koja sadrži C++ programski kod.

3.3.4. Squish

Radi postizanja automatizacije testiranja Squish [36] alat za testiranje sučelja Qt aplikacija nudi umetanje naredbi nad aplikacijom pod testom (engl. AUT, *Application Under Test*), uzimanje slika zaslona ili njegovog dijela, njihovo uspoređivanje s točkama verifikacije te ponovno izvođenje testova. Nad izrađenim slikama zaslona za verifikacijske točke mogu se kreirati maskiranja za postavljanje elemenata slike koji se pri testu moraju pojaviti na zaslonu testirane aplikacije ili oni koje ne želimo vidjeti na zaslonu. Također se unutar alata omogućava odabir jednog od načina usporedba slika verifikacijske točke i slike dobivene na testu [37]. Ponuđeni načini su:

- Strogo (engl. *Strict*) – ne nudi mogućnost podešavanja, odnosno očekuje potpuno podudaranje piksela obje slike u svim bojama, što nije često primjenjivo
- Piksel način (engl. *Pixel (fuzzy) mode*) – uspoređuje dvije slike po pikselima sa istim dimenzijama, a podržava mogućnost razlikovanja u postotcima od 0 do 100% čime se označava koliko piksela slike smije biti različito te u koliko smije biti razlika boje u pikselima (izračun razlike unutar spektra RGB)
- Histogram boja - uspoređivanje boja prema normaliziranom histogramu boja, primjenjivo za slike koje su rotirane, skalirane ili rotirane ali su im boje ostale iste ili se nisu promijenile, omogućava postavljanje rasporeda boja u košare (engl. *bins*) koji računaju koliko boja spada u koju kategoriju te postavljanje vrijednosti koja predstavlja dozvoljeni prag

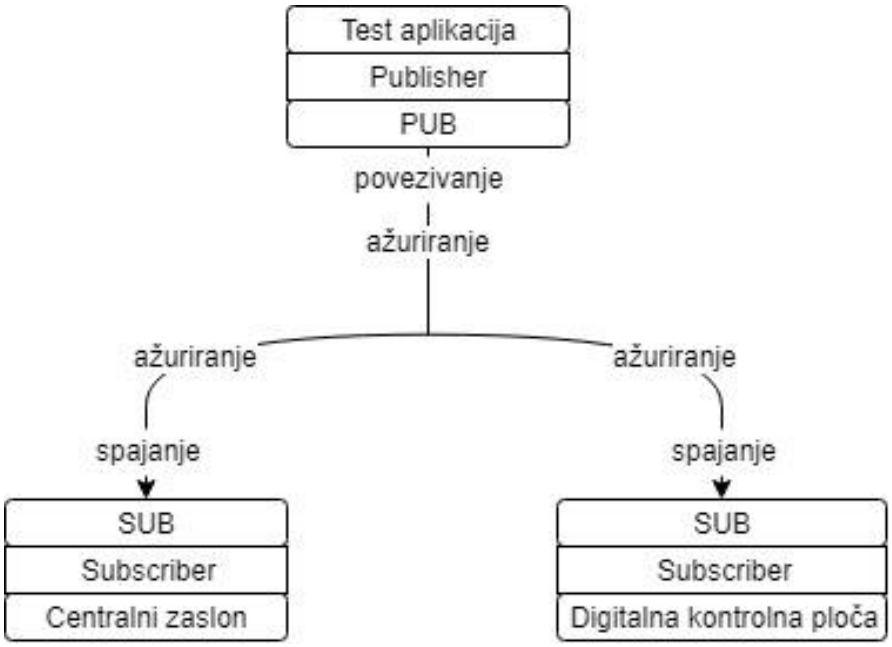
- Podudaranje (engl. *Correlation*) [38] – koristi sličnost statističkog koeficijenta podudaranja temeljenog na obradi signala, a omogućava postavljanje praga od 0% gdje ne postoji podudaranje do 100% gdje se nalazi savršeno podudaranje, a provodi se na razinama sive boje unutar slika koje su dobivene skaliranjem na istu veličinu te je neovisno o rezoluciji

3.3.5. ZeroMQ

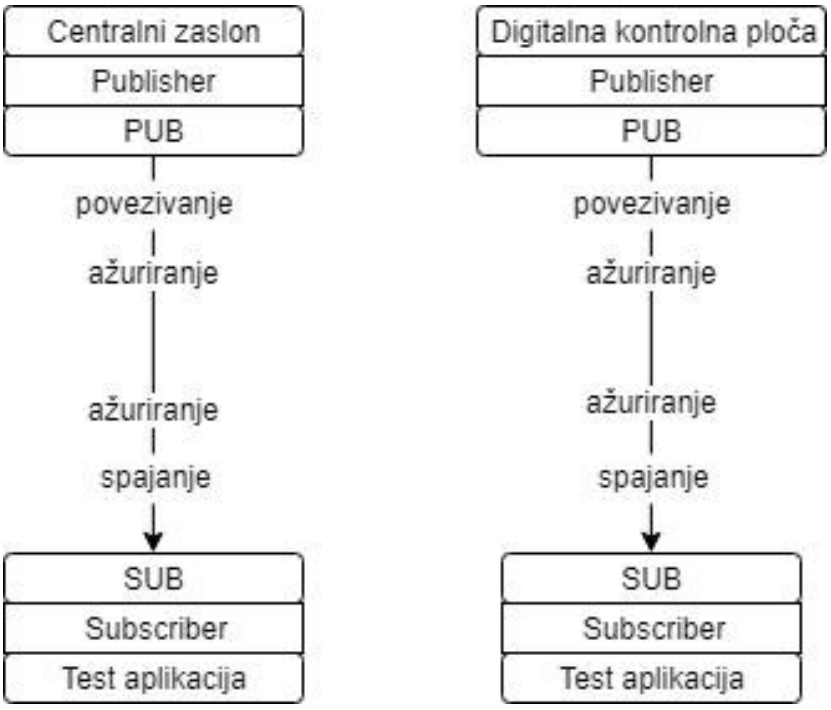
ZeroMQ [39] ili skraćeno ZMQ (odnosno ØMQ) je biblioteka namijenjena asinkronom prijenosu poruka visokim performansama. Razvijena je kroz besplatan program sa potpunom komercijalnom podrškom, te podržana od velike i aktivne zajednice koja radi na svima dostupnom kodu. Ciljevi su im postizanje nulte administracije, troška i otpada. Neke od mogućnosti korištenja ovakve biblioteke su:

- Spajanje programskih modula pisanih u bilo kojem programskom jeziku na bilo kojoj platformi
- Prijenos poruka preko inproc, IPC, TCP, TIPC, multicast
- Asinkroni I/O engine velike brzine u maloj biblioteci
- Podrška za sve moderne jezike i platforme
- Može se koristiti za izgradnju bilo koje arhitekture kao što su centralizirana, distribuirana, mala ili velika
- Pametni obrasci kao što su *pub-sub, push-pull i router-dealer*

Upravo se obrazac *pub-sub(publish-subscribe)* prema slici 3.11 koristi za komunikaciju između aplikacije za testiranje i zaslona vozila, te pruža neovisnost komunikacije između aplikacija izgrađenih u različitim inačicama aplikacije. Također takav se obrazac prema slici 3.12 koristi i za slanje povratne informacije natrag u aplikaciju za testiranje.



Slika 3.11. *Publish-subscribe komunikacija između aplikacije za testiranje i zaslona*



Slika 3.12. *Publish-subscribe komunikacija između zaslona i aplikacije za testiranje*

4. RAZVOJ APLIKACIJE I TESTIRANJE

U nastavku slijedi opis kreiranja aplikacije za simulaciju mehaničkih ulaza koja komunicira sa aplikacijama unutar vozila preko ZMQ komunikacije, dodavanje ZMQ komunikacije unutar aplikacija informacijsko-zabavnog sustava vozila, primjena QtTesta i QtQuickTesta unutar Qt Creatora za izradu testiranja jedinica te kreiranje verifikacijskih točaka i testova za integracijsko testiranje i testiranje sustava kroz alat Squish.

4.1. Razvoj aplikacije za simulaciju ulaza informacijsko-zabavnog sustava vozila

Za potrebe same mogućnosti automatiziranog testiranja grafičkog korisničkog sučelja aplikacija centralnog zaslona i digitalne nadzorne ploče osposobljena je aplikacija koja omogućava komunikaciju sa njima na način da se šalju signali preko ZMQ komunikacije na ulaze aplikacija identični onima sa mehaničkih ulaza. Pri tome mapirane su naredbe koje se šalju iz aplikacije koja simulira ulaze prema aplikacijama na zaslonima prema slici 4.1.

```
#ifndef IVIKNOBSMETHODMAPPING_H
#define IVIKNOBSMETHODMAPPING_H

...
#define SET_MODE_TORQUE_VECTORING "setModeTorqueVectoring"
...

...
#endif // IVIKNOBSMETHODMAPPING_H
```

Sl. 4.1. *Mapiranje naredbe za slanje preko ZMQ komunikacije*

Prema slici 4.2 prikazano je zaglavlje iz kojega je deklarirana klasa TestingIviKnobsApp koja implementira ZMQ sučelje sa svim potrebnim bibliotekama te sadrži sve deklaracije uključujući eksplicitni konstruktor koji u argumentu sadržava putanju do konfiguracijske datoteke za ZMQ *publish-subscribe* komunikaciju, podatke o *publisheru* i *subscriberu* kao što su IP adresa, port te tema koju se objavljuje i očekuje, virtualni destruktor, metodu za kreiranje objekta, Q_PROPERTY sa pripadajućim globalnim varijablama i metodama za čitanje i pisanje u članove koje su potrebni za komunikaciju između klase napisane u C++ programskom jeziku i QML sučelja aplikacije, a navedeni su kao Q_INVOKABLE što im omogućuje pozivanje iz QML .

```
/**
* @file       TestingIviKnobsApp.h
* @author     Leo Matančić
* @brief      Class for automated testing of IVI with knobs
*/

#ifndef TESTINGIVIKNOBSAPP_H
#define TESTINGIVIKNOBSAPP_H
#include <QObject>
#include <QDir>
#include "ZMQAdapter.h"
#include "Utils.h"

class TestingIviKnobsApp : public QObject
{
    Q_OBJECT
...
    Q_PROPERTY(int modeTorqueVectoringTest      MEMBER m_modeTorqueVectoringTest
    READ modeTorqueVectoringTest      WRITE setModeTorqueVectoringTest    )
...
protected:
    explicit TestingIviKnobsApp(const QString &zmqSettingsFile);
public:
   static TestingIviKnobsApp* getInstance(const QString &zmqSettingsFile = QDir::
currentPath()+ c_zmqSettingsPath + "zmqTestAppSettings.ini");

    virtual ~TestingIviKnobsApp();
...
    Q_INVOKABLE void setModeTorqueVectoringTest    (int modeTorqueVectoringTest);
    int modeTorqueVectoringTest             (){return m_modeTorqueVectoringTest;}
...
private:
    RA::ZMQAdapter m_zmqInterface;

...

    int m_modeTorqueVectoringTest;
...
};
#endif // TESTINGIVIKNOBSAPP_H
```

Slika 4.2. *Zaglavlje klase TestingIviKnobsApp*

U izvornom kodu prema slici 4.3 prikazane su implementacije metode zaglavlja, uključeno je zaglavlje koje mapira ZMQ poruke, u konstruktoru se pojavljuje inicijalizacija člana koji sadrži argument putanje do konfiguracijske datoteke vezane za ZMQ komunikaciju. Također, prikazana je metoda kojoj se pozivom iz QML sučelja i predajom argumenta postavlja vrijednost za Torque Vectoring funkcionalnost vozila preko simuliranog mehaničkog ulaza te se slaže poruka koja će biti poslana u QVariant tipu koji prihvaća veliku većinu Qt tipova, a ista poruka se rasčlanjuje na ulazu u aplikacije vozila. Analogno ovoj metodi definiraju se i metode za ostale funkcionalnosti unutar vozila, a koje su upravljane mehaničkim ulazima.

```
/**
* @file       TestingIviKnobsApp.cpp
* @author     Leo Matančić
* @brief      Class for automated testing of IVI with knobs
*/
#include "TestingIviKnobsApp.h"
#include "IviKnobsMethodMapping.h"
#include <QDebug>

TestingIviKnobsApp::TestingIviKnobsApp(const QString &zmqSettingsFile):
    m_zmqInterface(zmqSettingsFile)
{
    connect(&m_zmqInterface, &RA::ZMQAdapter::newMessage,
            this,&TestingIviKnobsApp::manageZMQMessage);
}

TestingIviKnobsApp *TestingIviKnobsApp::getInstance(const QString &zmqSettingsFile
)
{
    static TestingIviKnobsApp* instance = nullptr;

    if (instance == nullptr)
    {
        instance = new TestingIviKnobsApp(zmqSettingsFile);
    }

    return instance;
}

TestingIviKnobsApp::~TestingIviKnobsApp()
{

}
...

void TestingIviKnobsApp::setModeTorqueVectoringTest(int modeTorqueVectoringTest)
{
    m_modeTorqueVectoringTest = modeTorqueVectoringTest;
    qDebug()<<"setModeTorqueVectoringTest test value: "<<modeTorqueVectoringTest;
    RA::ZMQMessage message = RA::ZMQMessage(SET_MODE_TORQUE_VECTORING,QVariant::fr
omValue(modeTorqueVectoringTest));
    m_zmqInterface.sendMessage(message);
}
...
}
```

Slika 4.3. *Izvorni kod klase TestingIviKnobsApp*

Samo pokretanje aplikacije odvija se iz izvorne datoteke koja sadrži glavnu metodu main prema slici 4.4 te ona kreira objekt klase TestingIviKnobsApp i povezuje C++ programski kod sa QML sučeljem predajom pokazivača na objekt i postavljanjem njegovog konteksta unutar QML sučelja na „testingIviKnobsApp“ odnosno objekta koji se koristi za pozivanje metoda unutar Q_PROPERTY-ja C++ izvornog koda iz QML sučelja, a koje su označene sa Q_INVOKABLE.

```
#include <QGuiApplication>
#include <QQmlApplicationEngine>
#include <QQmlEngine>
#include <QQmlContext>

#include <QDebug>
#include <QDir>
#include "Utils.h"

#include "TestingIviKnobsApp.h"

int main(int argc, char *argv[])
{
    qputenv("QT_IM_MODULE", QByteArray("qtvirtualkeyboard"));

    if (qEnvironmentVariableIsEmpty("QTGLESSTREAM_DISPLAY")) {
        qputenv("QT_QPA_EGLFS_PHYSICAL_WIDTH", QByteArray("213"));
        qputenv("QT_QPA_EGLFS_PHYSICAL_HEIGHT", QByteArray("120"));

        QCoreApplication::setAttribute(Qt::AA_EnableHighDpiScaling);
    }
    QGuiApplication app(argc, argv);
    QQmlApplicationEngine engine;

    TestingIviKnobsApp *m_tikaPtr = TestingIviKnobsApp::getInstance();
    engine.rootContext()->setContextProperty("testingIviKnobsApp",m_tikaPtr);

    const QUrl url(QStringLiteral("qrc:/main.qml"));
    QObject::connect(&engine, &QQmlApplicationEngine::objectCreated,
                     &app, [url](QObject *obj, const QUrl &objUrl) {
        if (!obj && url == objUrl)
            QCoreApplication::exit(-1);
    }, Qt::QueuedConnection);
    engine.load(url);

    return app.exec();

}
```

Slika 4.4. *Pokretanje aplikacije iz izvornog koda main.cpp*

Programski kod sučelja prema slici 4.5 prikazuje izgradnju sučelja aplikacije sa svim svojim svojstvima i pozadine, postavljanje unaprijed definiranih početnih postavki sučelja te kreiranje prilagođenih elemenata sa njihovim svojstvima koja su ovisna o trenutnom izboru , njihovih akcija na ulazne kontrole mišem, promjenu stanja sa pozivanjem metoda iz C++ izvornog koda uz odgovarajuće argumente vezane za Torque Vectoring funkcionalnost simuliranu kroz sučelje ove aplikacije. Analogno ovoj funkcionalnosti kreirani su i ostali elementi grafičkog sučelja ove aplikacije.

```
import QtQuick 2.12
import QtQuick.Window 2.12
import QtQuick.Controls 2.12
import QtQuick.Controls.Styles 1.4

Window {
    id: window
    visible: true

    x:815
    y:0

    width: 550
    height: 1080

    maximumHeight: height
    maximumWidth: width

    minimumHeight: height
    minimumWidth: width

    title: qsTr("IVI GUI Testing")

    Component.onCompleted:
    {
        volumeSlider.value = 1;
        volumeSliderMouseArea.volumeMute = false;

        leftWindowPercentLevel.value = 1;
        rightWindowPercentLevel.value = 1;

        modeDrivetrainCustomDial.modeDrivetrain = 0;
        modeDrivetrainCustomDial.setStateModeDrivetrain();

        modeDriveCustomDial.modeDrive = 0;
        modeDriveCustomDial.setStateModeDrive();

        modeTorqueVectoringCustomDial.modeTorqueVectoring = 2;
        modeTorqueVectoringCustomDial.setStateModeTorqueVectoring();

        modeKersCustomDial.modeKers = 4;
        modeKersCustomDial.setStateModeKers();

        modeGearboxCustomDial.modeGearbox = 0;
        modeGearboxCustomDial.setStateModeGearbox();

        parkingButton.parkingBrakeValue = false;
    }
    Image {
        id: backgroundImage
        x: 0
        y: 0
        width: 550
        height: 1080
        visible: true
        fillMode: Image.PreserveAspectFit
        source: "qrc:/Resources/diplomski_gui_edited.png"
    }
```

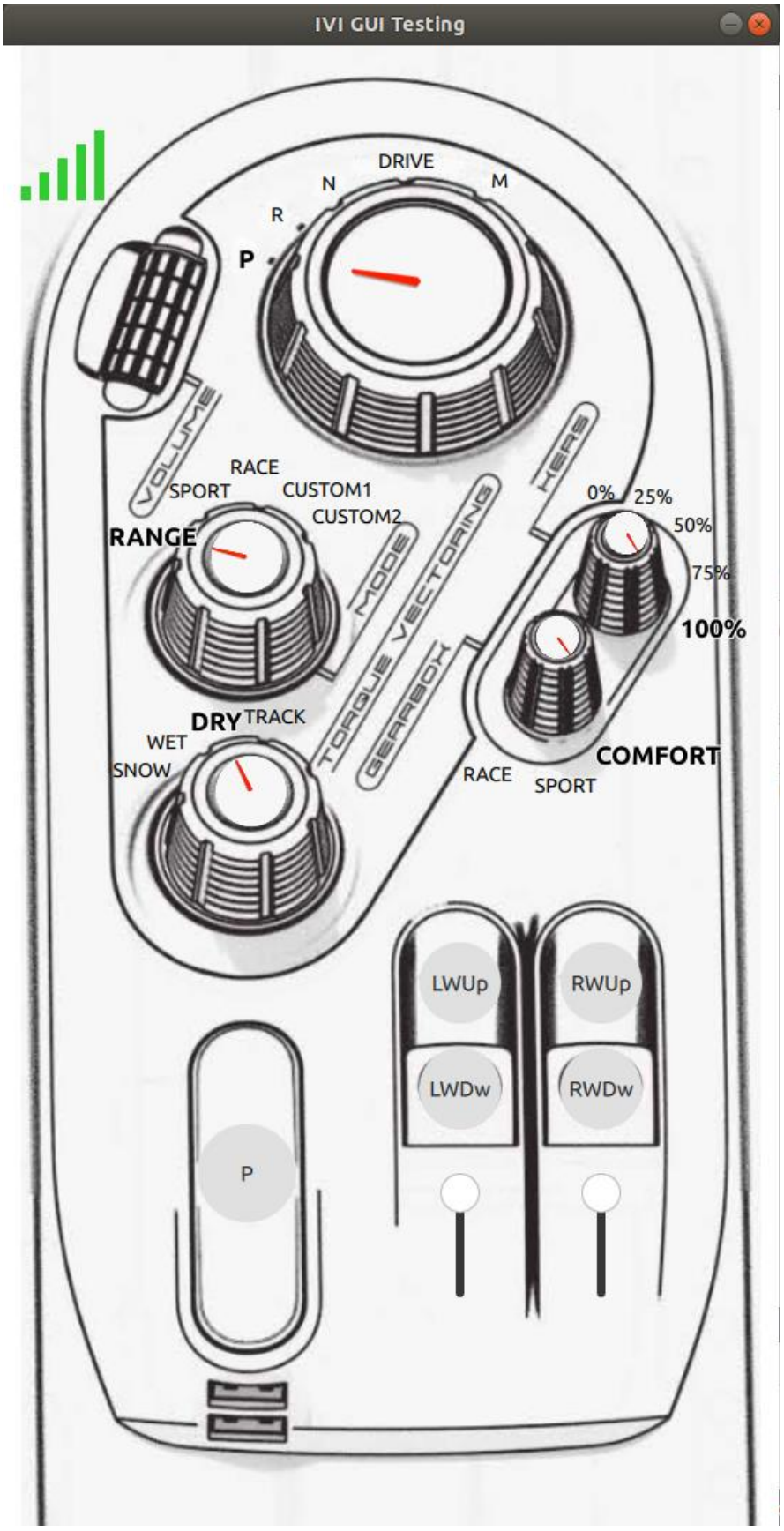
```
...
    Image
    {
        id:modeTorqueVectoringCustomDial
        x:151
        y:504
        width: 50
        height: 50
        visible: true
        source: "qrc:/Resources/custom_dial_small.png"

        property int modeTorqueVectoring;

        function setStateModeTorqueVectoring()
        {
         switch(modeTorqueVectoringCustomDial.modeTorqueVectoring.toString()){
                case "0":
                    modeTorqueVectoringCustomDial.state="snow"
                break;

                case "1":
                    modeTorqueVectoringCustomDial.state="wet"
                break;
...
      }
//possible states for torqueVectoring
        states: [
            State {
                name: "snow"
                PropertyChanges {

                    target: modeTorqueVectoringCustomDial
                    rotation: 15
                    modeTorqueVectoring: 0
                }
            },
            State {
                name: "wet"
                PropertyChanges {
                    target: modeTorqueVectoringCustomDial
                    rotation: 35
                    modeTorqueVectoring: 1
                }
            },
...
      }
onWheel:
            {
                if(wheel.angleDelta.y > 0)
                {
                    switch(modeTorqueVectoringCustomDial.state)
                    {
                        case "snow":
                            modeTorqueVectoringSnow.state = ""
                            //Test switch for wet torque vectoring mode
                            modeTorqueVectoringCustomDial.state = "wet"
                            modeTorqueVectoringWet.state = "selected"
                        break;
```

```
                        case "wet":
                            modeTorqueVectoringWet.state = ""
                            //Test switch for dry torque vectoring mode
                            modeTorqueVectoringCustomDial.state = "dry"
                            modeTorqueVectoringDry.state = "selected"
                        break;
...
            }

                       if(wheel.angleDelta.y < 0)

                    {

...
                         case "dry":
                            modeTorqueVectoringDry.state = ""
                             //Test switch for wet torque vectoring mode
                            modeTorqueVectoringCustomDial.state = "wet"
                            modeTorqueVectoringWet.state = "selected"
                        break;
                        case "wet":
                            modeTorqueVectoringWet.state = ""
                            //Test switch for snow torque vectoring mode
                            modeTorqueVectoringCustomDial.state = "snow"
                            modeTorqueVectoringSnow.state = "selected"
                        break;
                    }
                }
            }
        }
    }
       Text {
         id: modeTorqueVectoringSnow
         x: 78
         y: 505
         text: qsTr("SNOW")

         font.pixelSize: modeTorqueVectoringSnow.state == "selected" ? 18 : 14
         font.bold: modeTorqueVectoringSnow.state == "selected"
         style: modeTorqueVectoringSnow.state == "selected" ? Text.Outline : Text.
Normal; styleColor: "#f5f5f5"

         state: modeTorqueVectoringCustomDial.state == "snow" ? "selected" : ""

         states: [
             State {
                 name: "selected"
                 PropertyChanges {
                     target: modeTorqueVectoringSnow
                 }
             }
         ]
    }
       Text {
         id: modeTorqueVectoringWet
         x: 102
         y: 485
         text: qsTr("WET")
```

```
        font.pixelSize: modeTorqueVectoringWet.state == "selected" ? 18 : 14
        font.bold: modeTorqueVectoringWet.state == "selected"
        style: modeTorqueVectoringWet.state == "selected" ? Text.Outline : Text.N
ormal; styleColor: "#f5f5f5"
       state: modeTorqueVectoringCustomDial.state == "wet" ? "selected" : ""

        states: [
            State {
                name: "selected"
                PropertyChanges {
                    target: modeTorqueVectoringWet
                }
            }
        ]
    }

...

}
```

Slika 4.5. *QML sučelje aplikacije za simulaciju mehaničkih ulaza*

Prikaz izgleda sučelja ove aplikacije moguće je vidjeti prema slici 4.6 Ono sadrži mogućnost podešavanja zvuka, promjenu stupnja mjenjača, načina vožnje, vektorskog prijenosa okretnog momenta na kotače, namještanje postotka rekuperacije energije u baterije koristeći kinetičku energiju, podešavanje mjenjača, uporabu elektroničke parkirne kočnice te podizanje i spuštanje stakala lijeve i desne strane vozačeve kabine.



Slika 4.6. *Sučelje aplikacije koja simulira mehaničke ulaze vozila Rimac Concept_One*

4.2. Implementacija ZMQ sučelja unutar vozila

Nadalje implementirana je ZMQ biblioteka unutar aplikacije centralnog zaslona zaglavlja na slici 4.7 i izvornog koda na slici 4.8 kako bi se omogućila komunikacija sa aplikacijom koja simulira mehaničke ulaze vozila prisluškujući komunikaciju *publishera*. Također napravljena je konekcija između signala i utora kako bi se na novu primljenu poruku okinulo upravljanje dospjelom porukom i odradila određena funkcionalnost na sučelju pod slikom 4.9., a za koju je potrebno kontekstno postaviti kontekstno svojstvo sa objektom koji prosljeđuje podatke prema korisničkom sučelju. Sukladno tome odrađena je implementacija i za digitalnu nadzornu ploču za njezine odgovarajuće funkcionalnosti.

```
#ifndef CENTRALDISPLAY_H
#define CENTRALDISPLAY_H
...
#include "KnobsTestDataProvider.h"
#include "Utils.h"
class CentralDisplay :  ...
{
...
    Q_INVOKABLE void modeDriveTest(int modeDrive);
...
    void initializeZMQCommunicationHandling();  //ZMQ connects for messages
    //ZMQ required for testing
    ZMQAdapter m_zmqInterface;
    //Class required for knobs testing
    KnobsTestDataProvider *m_ktdpPtr;
...
}
#endif // CENTRALDISPLAY_H
```

Slika 4.7. *Zaglavlje klase aplikacije centralnog zaslona*

```
CentralDisplay::CentralDisplay(int argc, char *argv[],const QString &zmqSettingsF
ile) : ...
m_zmqInterface(zmqSettingsFile),  //managing ZMQ message from publisher
m_ktdpPtr(KnobsTestDataProvider::getInstance()),  //knobs test data provider  cla
ss with Q_PROPERTY values
...
{
...
       initializeZMQCommunicationHandling();  //ZMQ connects for messages
...
    m_view.setSource(QUrl("qrc:/resources/s1200/qml/screens/main.qml"));
    m_view.setColor(Qt::black);

#ifndef DESKTOP_APP
     m_view.setFlags( Qt::FramelessWindowHint );
#endif
    m_view.setX(2*1920);
    m_view.showFullScreen();
}
```

```
...
void CentralDisplay::manageZMQMessage(ZMQMessage message)
{
       QString requestedGuiChange = message.name();

       ...
    else if (requestedGuiChange == SET_MODE_TORQUE_VECTORING && message.value().t
ypeName() == QString("int"))
    {
        int modeTorqueVectoring;
        modeTorqueVectoring = message.value().toInt();
        m_ktdpPtr->setModeTorqueVectoring(modeTorqueVectoring);
    }
}
void CentralDisplay::initializeZMQCommunicationHandling()
{
    //managing ZMQ message from publisher
    connect(&m_zmqInterface, &RA::ZMQAdapter::newMessage,
            this,&CentralDisplay::manageZMQMessage);
}
void CentralDisplay::initializeQMLContext()
{
    QQmlContext *qmlCtx = m_view.engine()->rootContext();
    qmlCtx->setContextProperty("centralDisplayTest", this);
    qmlCtx->setContextProperty("knobsTestDataProvider", m_ktdpPtr);
...
}
```

Slika 4.8. *Izvorni kod klase aplikacije centralnog zaslona*

```
import QtQuick 2.5
import QtQuick.Controls 1.1
import QtMultimedia 5.0
import Rimac 1.0
import QtGraphicalEffects 1.0
...
Rectangle {
    id: container
    x: 60
    width: 1200
    height: 1920
    ...
   // ----- Start: Mechanical inputs ----------------------------- //
   Connections {
        target: knobsTestDataProvider
        ...
       onModeTorqueVectoringChanged: {

           if(modelPanels.count === 0 || modelPanels.get(0).idModel !== "torq
ueVectoring") {
               checkIfModelPanelsContain("torqueVectoring");
               addToListModel(3);
           }
```

```
            repeater.itemAt(0).value =  modeTorqueVectoring
            repeater.itemAt(0).show();
            centralDisplayTest.modeTorqueVectoringTest(modeTorqueVectoring);
        }
...
      function addToListModel(value) {
            switch (value) {
             case 3:

            modelPanels.insert( 0, {
            idModel: "torqueVectoring",
            fontCentar: 256,
            fontSides: 128,
            fontCentarMin: 192,
             title: "TORQUE VECTORING",
            modelBtn: [
                  {  value: 0, valueText: "SNOW"    , subText: ""},
                  {  value: 1, valueText: "WET"     , subText: ""},
                  {  value: 2, valueText: "DRY"     , subText: "" },
                              {  value: 3, valueText: "TRACK"   , subText: ""
             },
                        ]
            })
            break;
}
```

Slika 4.9. QML grafičko sučelje aplikacije centralnog zaslona

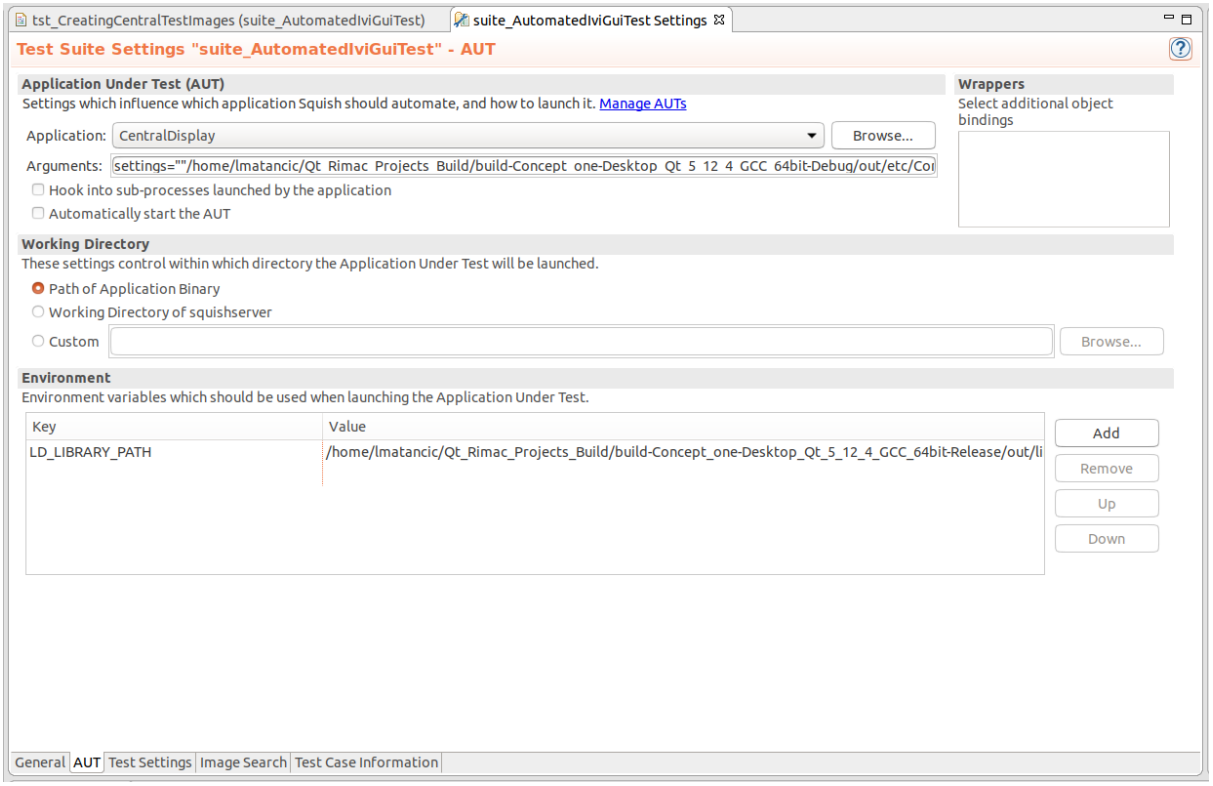
4.3. Implementacija testnog alata

4.3.1. Testiranje korištenjem Qt testiranja jedinica

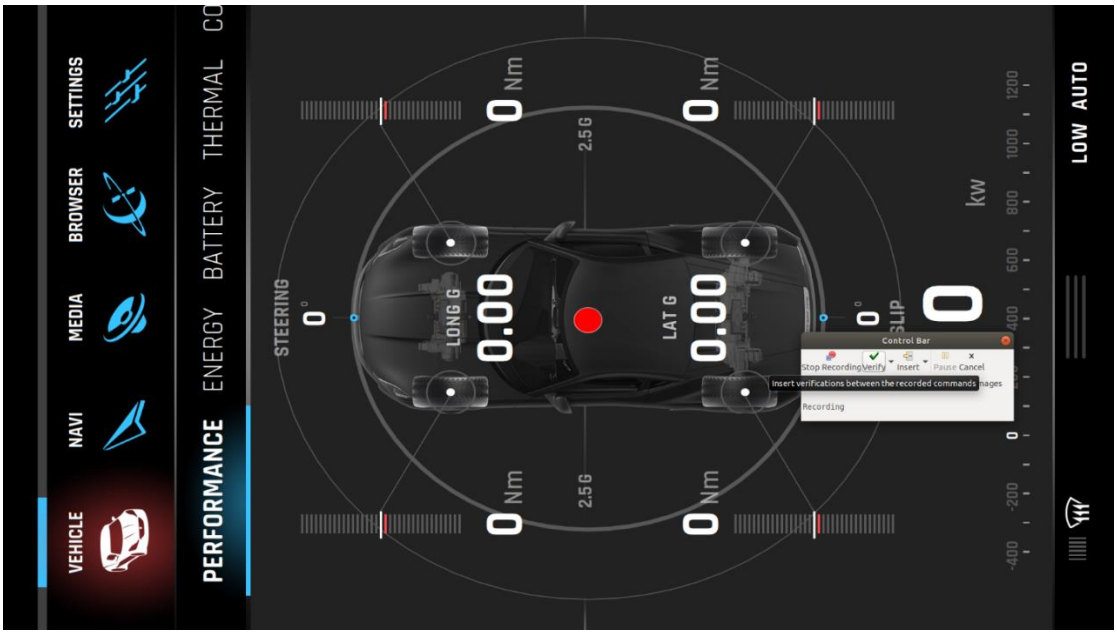
Integrirano razvojno okruženje Qt Creator doprinijelo je razvoju testova za izvorne kodove unutar C++ kroz QtTest *framework* odnosno uz QtQuickTest za QML grafičko sučelje. Za automatizaciju upotrijebljen je model testiranja vođenog podatcima na način da su zadani ulazni podatci prema aplikacijama centralnog zaslona i digitalne nadzorne ploče te se u odgovarajućoj metodi oba načina testiranja provjere izlazni podatci sa očekivanima. Nakon provođenja testa dobiva se izlazni podatak o prolasku ili padu pojedinih testova unutar korisničkog sučelja samog Qt Creatora te na samom kraju ispisa dobije se podatak o ukupnom broju izvedenih testova, testova koji su prošli te pali za određenu očekivanu izlaznu vrijednost. Pri tome bitno je napomenuti da je ovo sustav bijele kutije u kojem je potrebno ispravno definirati očekivane vrijednosti za zadane ulazne vrijednosti.

4.3.2. Testiranje korištenjem alata Squish

Pri izradi testova grafičkih korisničkih sučelja unutar alata Squish polazna točka za izradu testova je kreiranje testnog paketa i testnih slučajeva. Unutar testnog paketa potrebno je postaviti aplikacije pod testom prema slici 4.10 gdje se postavlja putanja do izgrađene *release* verzije Qt aplikacije te potrebni argumenti za pokretanje aplikacije i okruženje sa bibliotekama potrebno za aplikaciju koja nad kojom se izvodi test. Zatim se pokreće aplikacija pod testom te se dobiva mogućnost snimanja te odabira željenog načina verifikacije prema slici 4.11 što je u ovom slučaju uzimanje snimke sa zaslona aplikacije i hvatanje određenog svojstva sa samog sučelja te njegova verifikacije. Nakon što se uzme snimka zaslona ili njegovog određenog dijela kreira se verifikacijska točka koja se koristi za daljnju automatizaciju testova. Za izvršavanje testova potrebno je napisati programski kod koji će služiti u svrhu mogućnosti pokretanja više aplikacija prilikom testiranja za potrebe integracijskog testiranja ili testiranja sustava u kojem se osim pokretanja više testova pokreće i više skupova testova te kako bi se u točno vrijeme uhvatila slika zaslona i usporedila s slikom verifikacijske točke.

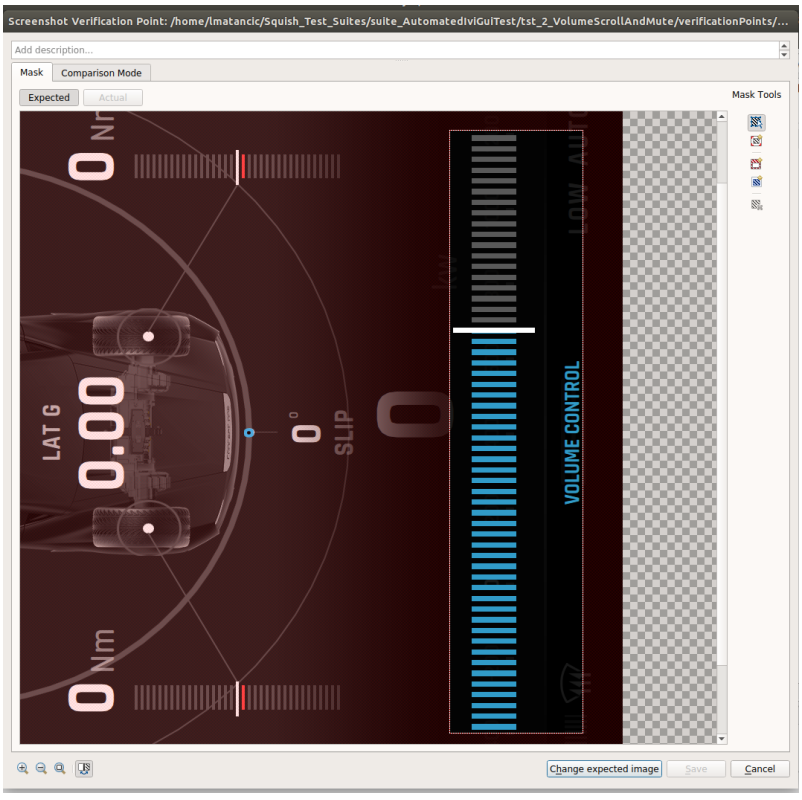


Slika 4.10. *Postavke aplikacije pod testom unutar Squish alata*



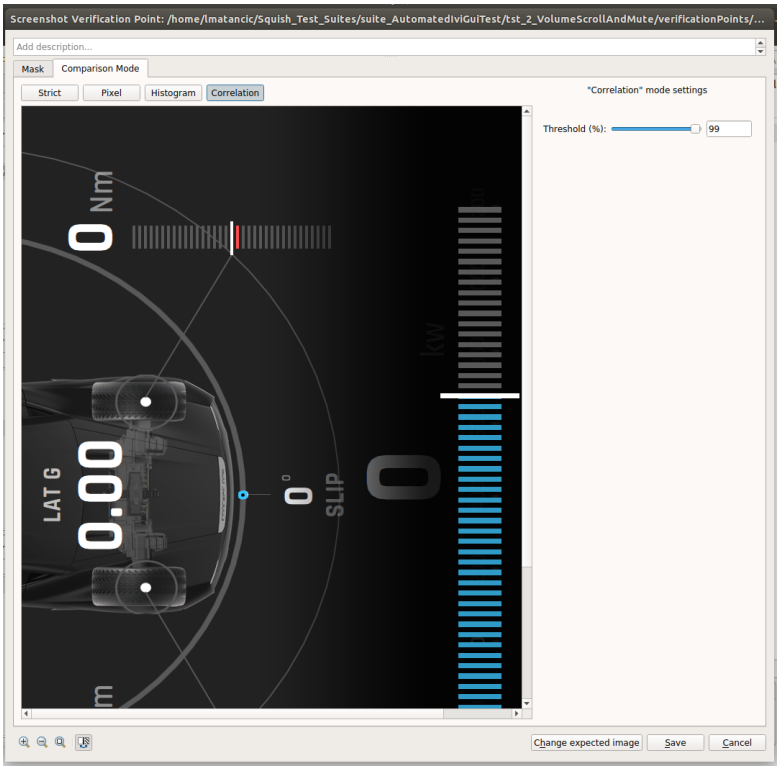
Slika 4.11. *Kreiranje verifikacijske točke unutar Squish alata*

Na dobivenu snimku zaslona za verifikacijsku točku može se dodati i maskiranje elemenata prema slici 4.12 na kojoj je prikazana pozitivna maska odnosno taj se element slike grafičkog korisničkog sučelja uzima u obzir s snimke zaslona i uspoređuje na testu ili ako možemo postaviti negativnu masku ako ne želimo da se taj element slike pojavi pri testiranju.



Slika 4.12. *Postavljanje pozitivne verifikacijske maske na verifikacijskoj točki*

U istom izborniku pod drugom karticom nalazi i način uspoređivanja slike uzete pri testu i slike dobivene verifikacijskom točkom. Kao zadovoljavajuća opcija načina usporedbe odabrana je metoda podudaranja (korelacije) slika sa pragom podudaranja od 99% prema slici 4.13.



Slika 4.13. *Postavljanje načina usporedbe slika na način podudaranja*

4.4. Pisanje testnih slučajeva

U ovom potpoglavlju prikazani su pojedini primjeri testiranja jedinica, integracijskog testiranja i testiranja sustava uz priloženi programski kod. Testiranje jedinica napravljeno je kroz Qt Creator, dok se za integracijsko i sustavsko testiranje koristi Squish.

4.4.1. Testiranje jedinica

Testiranje jedinica prema slici 4.14 prikazuje testiranje C++ jedinice koda koja prima raščlanjenu vrijednost primljene ZMQ poruke te ju postavlja pomoću metode koja prima argument sa željenom vrijednošću gdje je u metodi koja u nazivu iza imena test metode ima dodatak „_data“ koji označava kreiranje tablice za ulazne i očekivane vrijednosti za testiranje unutar raspona, na njegovim granicama te van raspona te provjere kako ne bi došlo do postavljanja neželjene vrijednosti. Zatim se u metodi koja počinje sa „test_“ izvršava test koji kreira objekt metode koju aplikacija koristi, dohvaća vrijednosti, izvršava test te daje rezultate testiranja koristeći tablicu.

```
#include <QtTest>

// add necessary includes here
#include "KnobsTestDataProvider.h"

class CentralModeTorqueVectoringTestCpp : public QObject
{
    Q_OBJECT

public:
    CentralModeTorqueVectoringTestCpp();
    ~CentralModeTorqueVectoringTestCpp();

private slots:
    void test_setTorqueVectoring_data();
    void test_setTorqueVectoring();
};

CentralModeTorqueVectoringTestCpp::CentralModeTorqueVectoringTestCpp()
{ }

CentralModeTorqueVectoringTestCpp::~CentralModeTorqueVectoringTestCpp()
{ }

void CentralModeTorqueVectoringTestCpp::test_setTorqueVectoring_data()
{
    QTest::addColumn<int>("input");
    QTest::addColumn<int>("expected");

    QTest::newRow("Snow")                   << 0 << 0;
    QTest::newRow("Wet")                    << 1 << 1;
    QTest::newRow("Dry")                    << 2 << 2;
    QTest::newRow("Track")                  << 3 << 3;
    QTest::newRow("RandomPositiveValue")    << 100 << 3;
    QTest::newRow("RandomPositiveValue")    << 99999 << 3;
    QTest::newRow("Snow")                   << 0 << 0;
    QTest::newRow("RandomPositiveValue")    << 11 << 0;
    QTest::newRow("RandomNegativeValue")    << -9999 << 0;
    QTest::newRow("Dry")                    << 2 << 2;
    QTest::newRow("OverBoundValueTest")     << 4 << 2;
    QTest::newRow("Wet")                    << 1 << 1;
    QTest::newRow("UnderBoundValueTest")    << -1 << 1;
}
void CentralModeTorqueVectoringTestCpp::test_setTorqueVectoring()
{
    QFETCH(int,input);
    QFETCH(int,expected);

    KnobsTestDataProvider* m_ktdpPtr = KnobsTestDataProvider::getInstance();
    m_ktdpPtr->setModeTorqueVectoring(input);
     qDebug() << "Input value= " << input << ", Expected value= " << expected
<< ", Actual value= " << m_ktdpPtr->modeTorqueVectoring();
    QCOMPARE(m_ktdpPtr->modeTorqueVectoring(),expected);
}
QTEST_MAIN(CentralModeTorqueVectoringTestCpp)

#include "tst_centralmodetorquevectoringtestcpp.moc"
```

Slika 4.14. *Testiranje jedinice C++ funkcionalnosti Torque Vectoring*

Kod QML testiranja prema slici 4.15 prije samog pokretanja testa potrebno je znati da je aplikacija pokrenuta koristeći utor koji se okida na signalizaciju pokrenute aplikacije. Nadalje u slučaju korištenja Q_PROPERTYJA iz C++ programa mora se dodati svojstvo kontekstu aplikacije koje kreira objekt željene klase i daje mogućnost pozivanja njegovih metoda.

```
#include <QtQuickTest>
#include <QQmlEngine>
#include <QQmlContext>
#include <QQuickView>

#include "KnobsTestDataProvider.h"

class Setup : public QObject
{
    Q_OBJECT

public:
    Setup(){}
    ~Setup(){}

     KnobsTestDataProvider *m_ktdpPtr;

public slots:
    void qmlEngineAvailable(QQmlEngine *engine)
    {
        m_ktdpPtr = KnobsTestDataProvider::getInstance();
        engine->rootContext()-
>setContextProperty("knobsTestDataProvider", m_ktdpPtr);
    }
};

QUICK_TEST_MAIN_WITH_SETUP(main,Setup)
#include "main.moc"
```

Slika 4.15. *Glavna klasa potrebna za izvođenje QML testa*

Pri testiranju QML grafičkog sučelja također korištena je metoda automatizacije testiranja vođena podatcima kroz postavljenu tablicu sa zadanim ulaznim i očekivanim vrijednostima. Ova metoda koristi se kako bi se provjerila ispravnost svojstava grafičkog korisničkog sučelja, odnosno da je postavljena vrijednost odgovarajuća očekivanoj na zadani ulaz.

```
import QtTest 1.0

import QtQuick 2.5
import QtQuick.Controls 1.1
import QtMultimedia 5.0
import Rimac 1.0
import QtGraphicalEffects 1.0
...
```

```
Rectangle {
    id: container
    x: 60
    width: 1200
    height: 1920
    focus: true

    property int input_modeKers;
    property int expected_modeKers;
    property int actual_modeKers;

    ...

    Connections
    {

        target: knobsTestDataProvider
        ...

        onModeTorqueVectoringChanged:
        {
            if(modelPanels.count === 0 || modelPanels.get(0).idModel !== "torqueVe
ctoring")
            {
                checkIfModelPanelsContain("torqueVectoring");
                addToListModel(3);
            }
            repeater.itemAt(0).value =  modeTorqueVectoring
            repeater.itemAt(0).show();
        }
        ...
}

TestCase
    {
        name: "CentralModeKersTestQML"

        function test_setModeKersTest_data()
        {
            return [
                  {tag:"KersOff",             input : 0, expectedValue : 0},
                  {tag:"Kers25",              input : 1, expectedValue : 1},
                  {tag:"Kers50",              input : 2, expectedValue : 2},
                  {tag:"Kers75",              input : 3, expectedValue : 3},
                  {tag:"Kers100",             input : 4, expectedValue : 4},
                  {tag:"OverBoundValueTest",  input : 5, expectedValue : 4},
                  {tag:"UnderBoundValueTest", input : -1, expectedValue : 4},
                  {tag:"Kers25",              input : 1, expectedValue : 1},
                  {tag:"RandomPositiveValue", input : 685, expectedValue : 1},
                  {tag:"KersOff",             input : 0, expectedValue : 0},
                  {tag:"RandomNegativeValue", input : -685, expectedValue : 0},
                  {tag:"Kers75",              input : 3, expectedValue : 3}
            ];
        }
```

```
        function test_setModeKersTest(data)
        {
            container.input_modeKers = data.input;
            knobsTestDataProvider.setModeKers(container.input_modeKers);
            container.expected_modeKers = data.expectedValue;
            console.log("Input value= " + container.input_modeKers + ", Expected
value= " + container.expected_modeKers + ", Actual value= " + container.actual_mo
deKers);
            compare(container.actual_modeKers,container.expected_modeKers)
        }
    }
}
```

Slika 4.16. *Testiranje jedinice QML grafičkog sučelja funkcionalnosti Mode KERS*

4.4.2. Integracijsko testiranje

Integracijsko testiranje provedeno je unutar Squish alata za testiranje QML grafičkih sučelja. Kao metoda verifikacije korištena je metoda zasnovana na snimkama zaslona, a za usporedbu ispravnosti s verifikacijskom točkom korištena je metoda korelacije. Zatim je za svaku funkcionalnost napisan kod koji pokreće test i aplikacije pod testom te izvodi testiranje. Testiranje se provodi tako da se unutar aplikacije koja simulira mehaničke ulaze postavi pokazivač miša i odradi zadana funkcionalnost, a zatim se obavi verifikacijski test na zadanom zaslonu. Sami kod je modularan na način da se micanjem zakomentiranog dijela koda odabire koje aplikacije se koriste u integracijskom testu, a ujedno ga se može koristiti unutar sustavskog testiranja vraćajući kod unutar komentiranog područja jer slučaju sustavskog testiranja se koristi master skripta.

```
import * as names from 'names.js';

function main()
{
    var modeTorqueVectoring_Central = ["ModeTorqueVectoringCentral/ModeTorq
ueVectoringSnowCentral","ModeTorqueVectoringCentral/ModeTorqueVectoringWetC
entral","ModeTorqueVectoringCentral/ModeTorqueVectoringDryCentral","ModeTor
queVectoringCentral/ModeTorqueVectoringTrackCentral"]
    var modeTorqueVectoring_Cluster = ["ModeTorqueVectoringCluster/ModeTorq
ueVectoringSnowCluster","ModeTorqueVectoringCluster/ModeTorqueVectoringWetC
luster","ModeTorqueVectoringCluster/ModeTorqueVectoringDryCluster","ModeTor
queVectoringCluster/ModeTorqueVectoringTrackCluster"]

//ctx_Central = startApplication("CentralDisplay settings=\"\"/home/lmatanc
ic/Qt_Rimac_Projects_Build/build-Concept_one-Desktop_Qt_5_12_4_GCC_64bit-
Debug/out/etc/Concept_one/CentralDisplay_config/p10-central.ini\"\"");
```

```
//ctx_Cluster = startApplication("ClusterDisplay settings=\"\"/home/lmatancic/
Qt_Rimac_Projects_Build/build-Concept_one-Desktop_Qt_5_12_4_GCC_64bit-
Debug/out/etc/Concept_one/ClusterDisplay_config/p10-cluster.ini\"\"");
//    ctx_KnobsApp = startApplication("AutomatedIVITestforGUIApp");
//
//    snooze(2);

    setApplicationContext(ctx_KnobsApp);

    mouseClick(waitForImage("image5", { }, waitForObjectExists(names.iVIGUITes
tingQQuickWindowQmlImpl)));

    mouseWheel(waitForObject(names.iVIGUITestingModeTorqueVectoringCustomDialI
mage), 19, 23, 0, 15, Qt.NoModifier);
    //insert screenshot verifications on both screens here
    snooze(0.2);
    setApplicationContext(ctx_Central);
    test.vp(modeTorqueVectoring_Central[3]);
    setApplicationContext(ctx_Cluster);
    test.vp(modeTorqueVectoring_Cluster[3]);
    snooze(0.2);
    setApplicationContext(ctx_KnobsApp);

    for (i = 2; i >= 0; i--)
    {

        mouseWheel(waitForObject(names.iVIGUITestingModeTorqueVectoringCustomD
ialImage), 19, 23, 0, -15, Qt.NoModifier);
        //insert screenshot verifications on both screens here
        setApplicationContext(ctx_Central);
        test.vp(modeTorqueVectoring_Central[i]);
        setApplicationContext(ctx_Cluster);
        test.vp(modeTorqueVectoring_Cluster[i]);
        snooze(0.2);
        setApplicationContext(ctx_KnobsApp);
    }

    mouseWheel(waitForObject(names.iVIGUITestingModeTorqueVectoringCustomDialI
mage), 19, 23, 0, 15, Qt.NoModifier);
    //insert screenshot verifications on both screens here
    setApplicationContext(ctx_Central);
    test.vp(modeTorqueVectoring_Central[1]);
    setApplicationContext(ctx_Cluster);
    test.vp(modeTorqueVectoring_Cluster[1]);

    snooze(1);
}
```

Slika 4.17. *Integracijsko testiranje funkcionalnosti Torque Vectoring grafičkog korisničkog sučelja*

4.4.3. Sustavsko testiranje

Master skripta dio je radnog prostora testiranja zajedno sa integracijskim testovima u Squishu. Skripta definira sustavski test na način da postavlja varijable okruženja, pali sve zadane aplikacije te uzima testove i izvodi ih redoslijedom, preskačući one testove koje označimo unutar varijable „ignore“. Sustavski test izvodi se na funkcionalnoj i nefunkcionalnoj razini. Funkcionalnu razinu testa obuhvaća testiranje funkcionalnosti na sučelju dok nefunkcionalnu razinu čini zapis korištenja resursa pojedine aplikacije kod testiranja. Također pri izvođenju hvata moguće pogreške i ispisuje ih, a ima i funkcionalnost spremanja zapisa izvođenja testova.

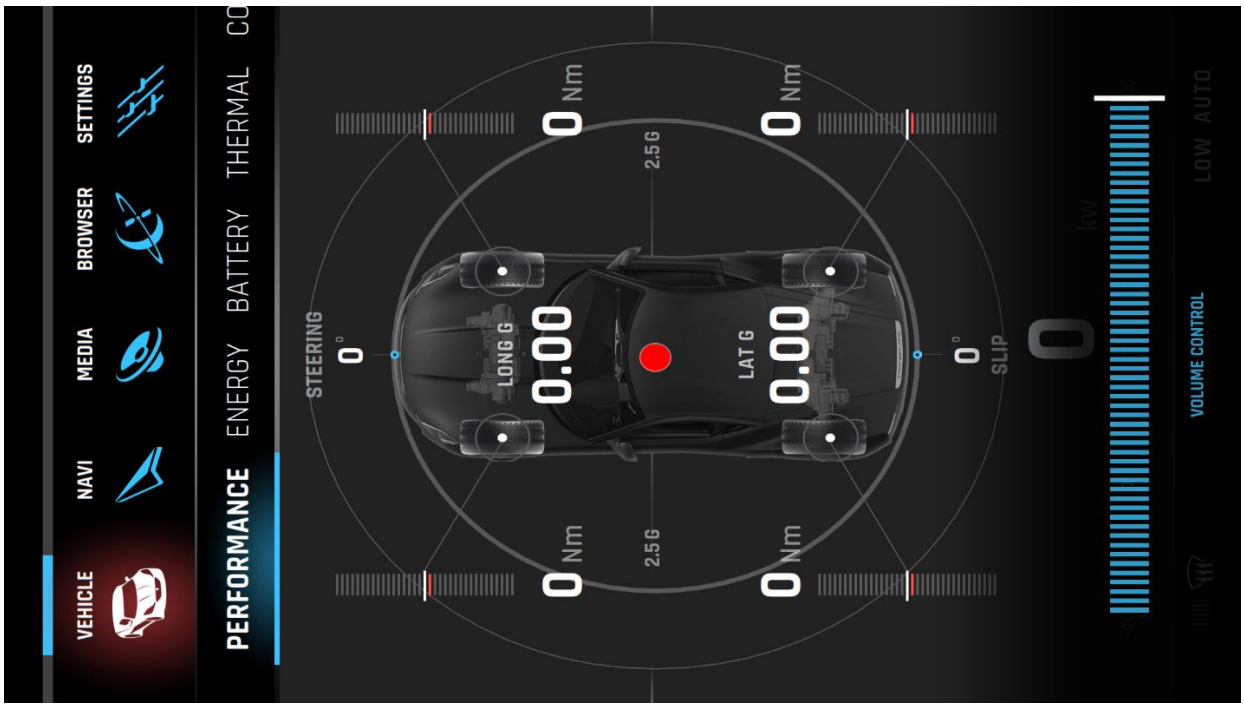
```
import * as names from 'names.js';

function main()
{
    LD_CONFIG_PATH="/home/lmatancic/Qt_Rimac_Projects_Build/build-Concept_one-
Desktop_Qt_5_12_4_GCC_64bit-
Release/out/lib;/home/lmatancic/Qt_Rimac_Projects_Build/build-
AutomatedIviGuiTestingApp-Desktop_Qt_5_12_0_GCC_64bit-Release/out/lib"

    var ignore = ["tst_CreatingCentralTestImages", "tst_CreatingClusterTestImages"
];
    var paths = OS.listDir("..");
    for (var i in paths) {
        var test_path = paths[i];
        if (inArray(ignore, test_path) ||
            test_path.indexOf("tst_") != 0 ||
            test_path == basename(OS.cwd())) {
            continue;
        }
        test.log("Executing: " + test_path);
        source("../" + test_path + "/test.js");

        // Start the application, if not running; useful
        // to ignore application crashes or application
        // having been stopped by previous test case:
        if (applicationContextList().length == 0) {

            ctx_Central = startApplication("CentralDisplay settings=\"\"/home/lmat
ancic/Qt_Rimac_Projects_Build/build-Concept_one-Desktop_Qt_5_12_4_GCC_64bit-
Debug/out/etc/Concept_one/CentralDisplay_config/p10-central.ini\"\"");
            ctx_Cluster = startApplication("ClusterDisplay settings=\"\"/home/lmat
ancic/Qt_Rimac_Projects_Build/build-Concept_one-Desktop_Qt_5_12_4_GCC_64bit-
Debug/out/etc/Concept_one/ClusterDisplay_config/p10-cluster.ini\"\"");
            ctx_KnobsApp = startApplication("AutomatedIVITestforGUIApp")
            snooze(2);
        }
        try {
            main(); // Executes the source'd test case's main() function
        } catch (e) {
            test.fatal("Error occurred in test case: " + test_path + ": " + e);
        }
    }
}
```

```
 function inArray(array, item)
{
    for (var i = 0; i < array.length; ++i) {
        if (array[i] == item)
            return true;
    }
    return false;
}

function basename(path)
{
    for (var i = path.length - 1; i >= 0; --i) {
        if (path[i] == "/" || path[i] == "\\") {
            return path.substring(i + 1);
        }
    }
    return path;
}
```

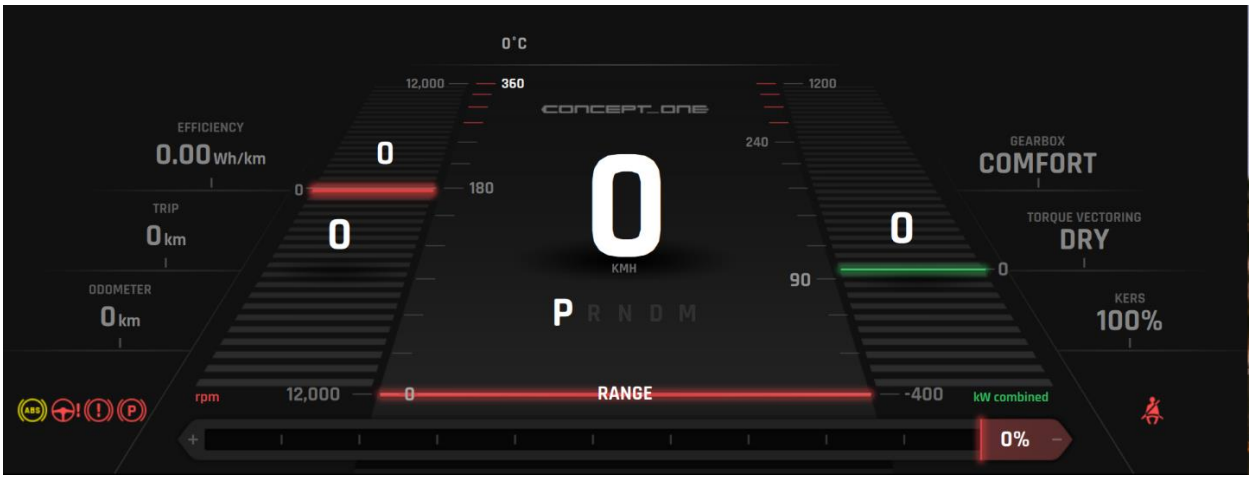
Slika 4.18. *Master skripta sustavskog testiranja*

4.5. Sučelje aplikacija koje se testiraju

Grafička sučelja aplikacija pod testom unutar Squish alata korištena za potrebe integracijskog i sustavskog testiranja grafičkog korisničkog sučelja vozila Rimac Concept_One prikazana su na slikama 4.19. i 4.20. Slike prikazuju veliku većinu trenutno testiranih funkcionalnosti grafičkog korisničkog sučelja upravljanog sa signalima mehaničkih ulaza.



Slika 4.19. *Centralni zaslon Rimac Concept_One*



Slika 4.20. *Digitalna nadzorna ploča Rimac Concept_One*

5. REZULTATI TESTIRANJA S ANALIZOM

U ovom poglavlju bit će prikazani provedeni testni slučajevi za sve testove jedinica, integracijsko testiranje te sustavsko testiranje.

5.1. Prikaz pokrivenih testnih slučajeva

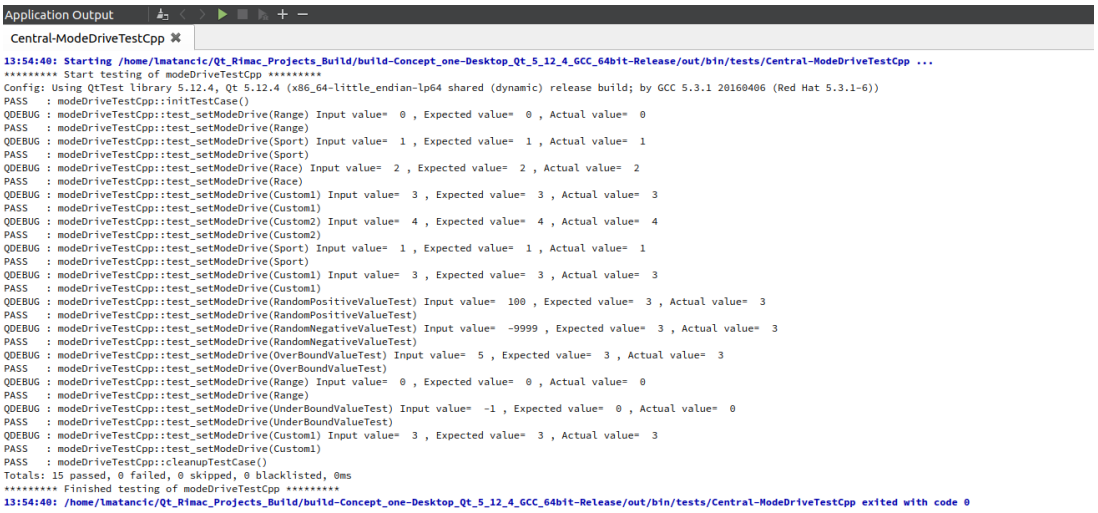
Prema tablici 5.1 prikazani su testni slučajevi pokriveni ovim automatiziranim sustavom testiranja ovisno o funkcionalnosti, zaslonu na kojem se obavlja testiranje te tip testiranja.

Tablica 5.1. *Pokriveni testni slučajevi*

1 - Centralni zaslon 2 - Digitalna nadzorna ploča 3 – 1 i 2 zajedno	Testiranje jedinica		Integracijsko testiranje	Sustavsko testiranje
	C++	QML		
Podešavanje zvuka	1/1	0/1	1/1	3/3
Stupanj mjenjača	1/3	2/3	3/3	3/3
Način vožnje	1/3	2/3	3/3	3/3
Vektorski prijenos okretnog momenta na kotače	1/3	2/3	3/3	3/3
Kinetičko obnavljanje energije	2/3	1/3	3/3	3/3
Podešavanje mjenjača	2/3	1/3	3/3	3/3
Elektronička parkirna kočnica	2/2	2/2	2/2	2/2

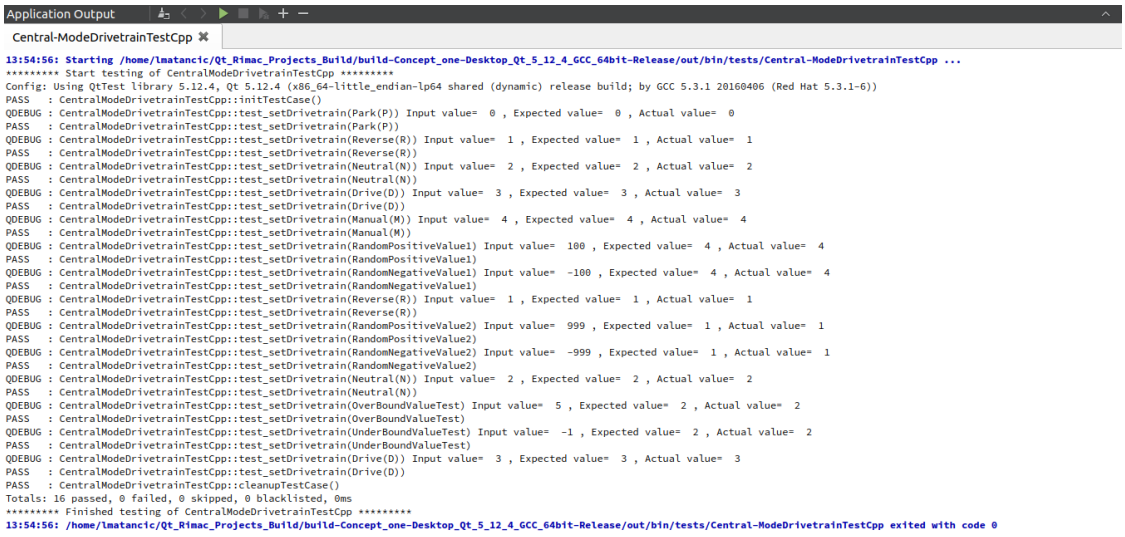
5.2. Rezultati testiranja jedinica

Prema slici 5.1 prikazan je rezultat testiranja C++ izvornog koda koristeći QtTest framework nad metodom ModeDrive aplikacije na centralnom zaslonu, a koja predstavlja funkcionalnost načina vožnje vozila. Provedeno je 15 testova promjene stanja unutar granica, na rubovima i van granica te su za sva stanja dobiveni očekivani rezultati, odnosno dobiveno je 15 prolaza na testu, 0 padova testa, 0 preskočenih testova te niti jedna funkcija se nije pronašla u crnoj listi testa, a trajanje izvođenja testa bilo je 0 milisekundi.



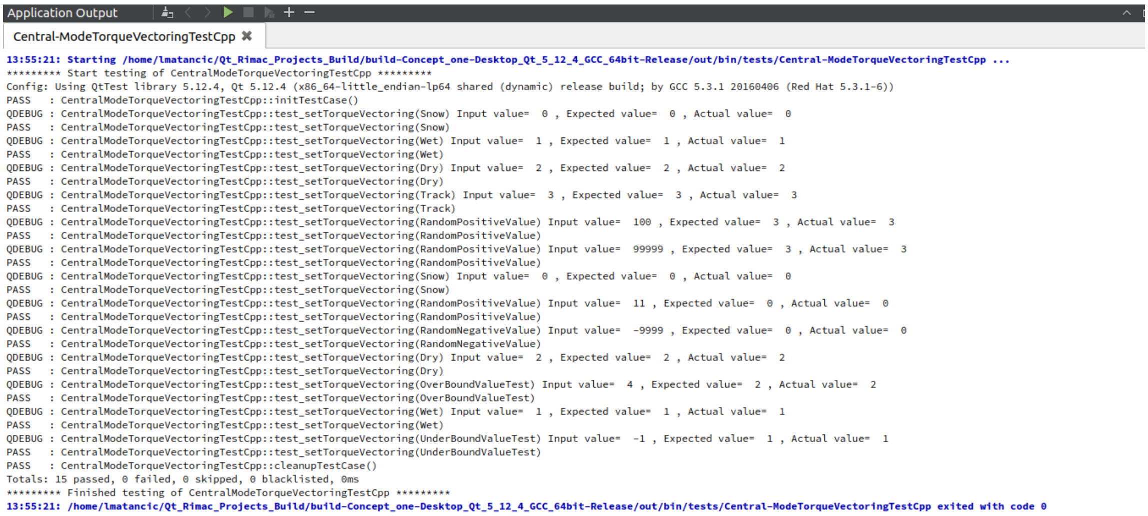
Slika 5.1. *Testiranje centralnog zaslona, QtTest test jedinice, metoda ModeDrive*

Iduća testirana metoda iz C++ izvornog koda aplikacije na centralnom zaslonu prema slici 5.2 je ModeDrivetrain koja predstavlja stupanj mjenjača, nad kojom je provedeno 16 testova i svi su uspješno prošli, a vrijeme izvođenja testa je 0 milisekundi.



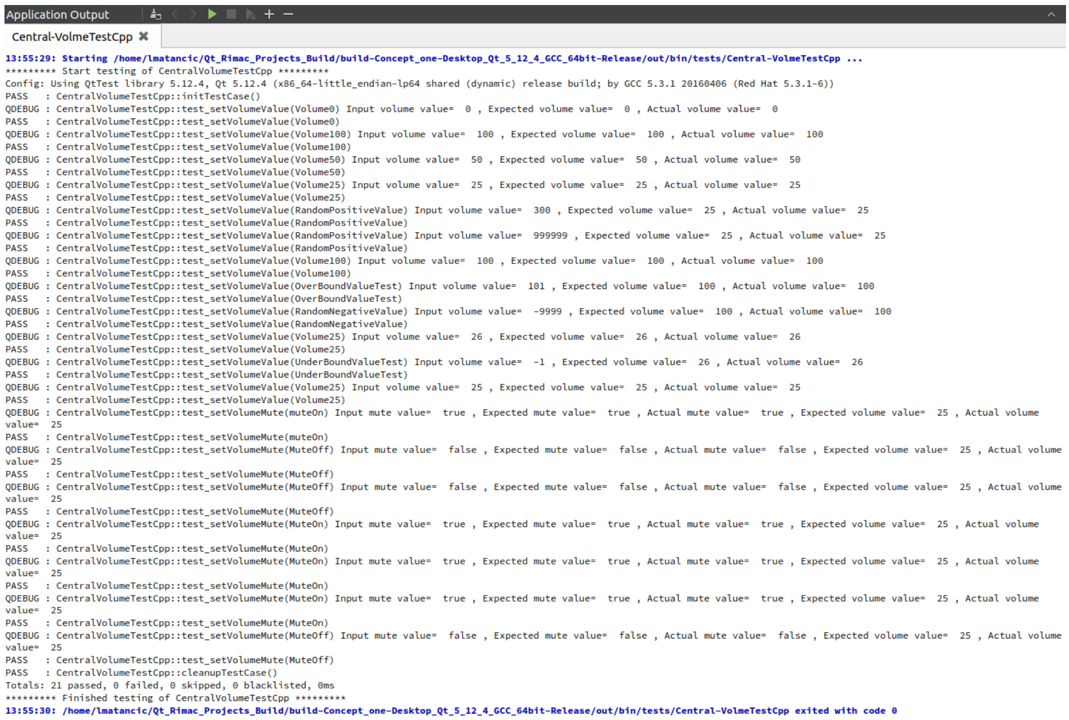
Slika 5.2. *Testiranje centralnog zaslona, QtTest test jedinice, metoda ModeDrivetrain*

Treći po redu test nad aplikacijom centralnog zaslona iz C++ izvornog koda je nad metodom ModeTorqueVectoring, koja se koristi za podešavanje vektorskog načina podešavanja okretnog momenta na kotačima. Odrađeno je 15 testova i svi su uspješno prošli, a vrijeme izvođenja testa bilo je 0 milisekundi.



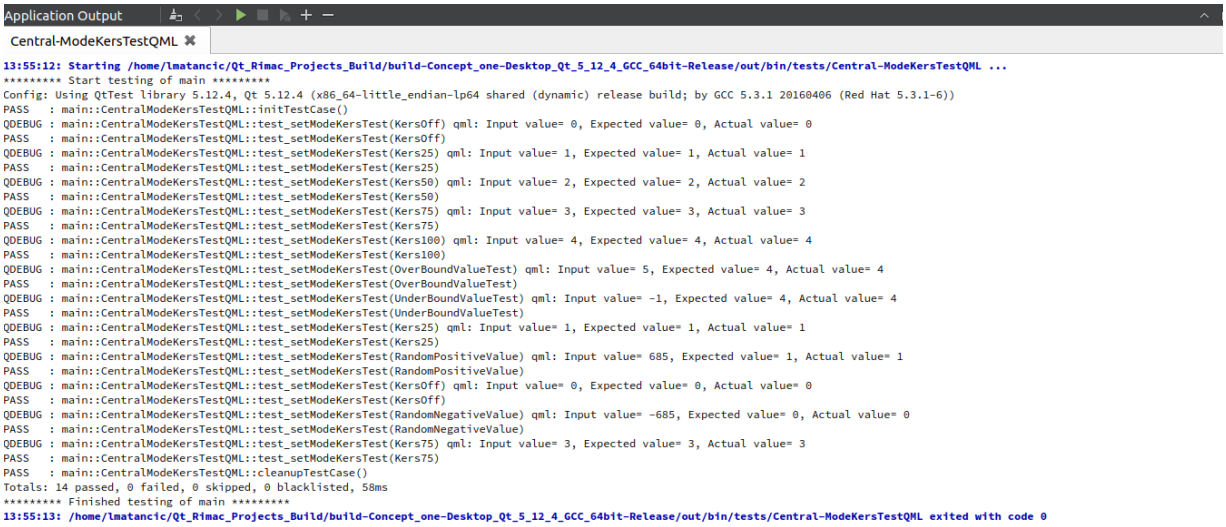
Slika 5.3. *Testiranje centralnog zaslona, QtTest test jedinice, metoda ModeTorqueVectoring*

Prema slici 5.4 prikazan je rezultat testiranja metoda setVolumeValue i setVolumeMute koje služe podešavanju i gašenju zvuka. Ukupno je proveden 21 test, a svi su rezultati bili odgovarajući očekivanima sa vremenom izvođenja testa 0 milisekundi.



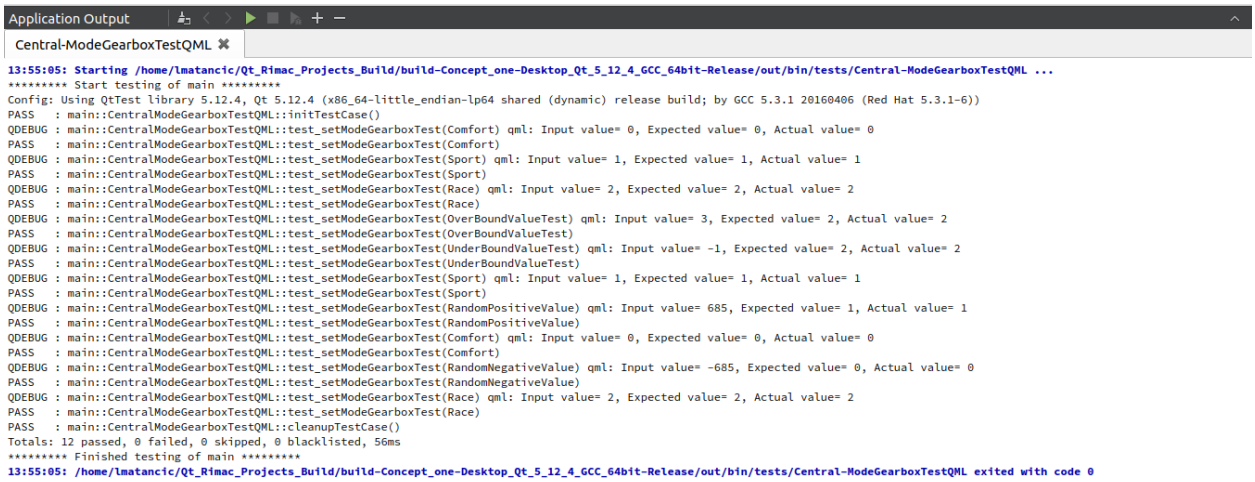
Slika 5.4. *Testiranje centralnog zaslona, QtTest test jedinice, metode setVolumeValue i setVolumeMute*

Zatim je nad centralnim zaslonom testirana metoda ModeKers koristeći QtQuickTest čiji je rezultat prikazan prema slici 5.5 čime se provjerava vrijednost svojstva grafičkog korisničkog sučelja u QML kodu nakon izvršene funkcije. Odrađeno je 14 testova, a sve vrijednosti odgovarale su očekivanima uz vrijeme trajanja testa od 58 milisekundi.



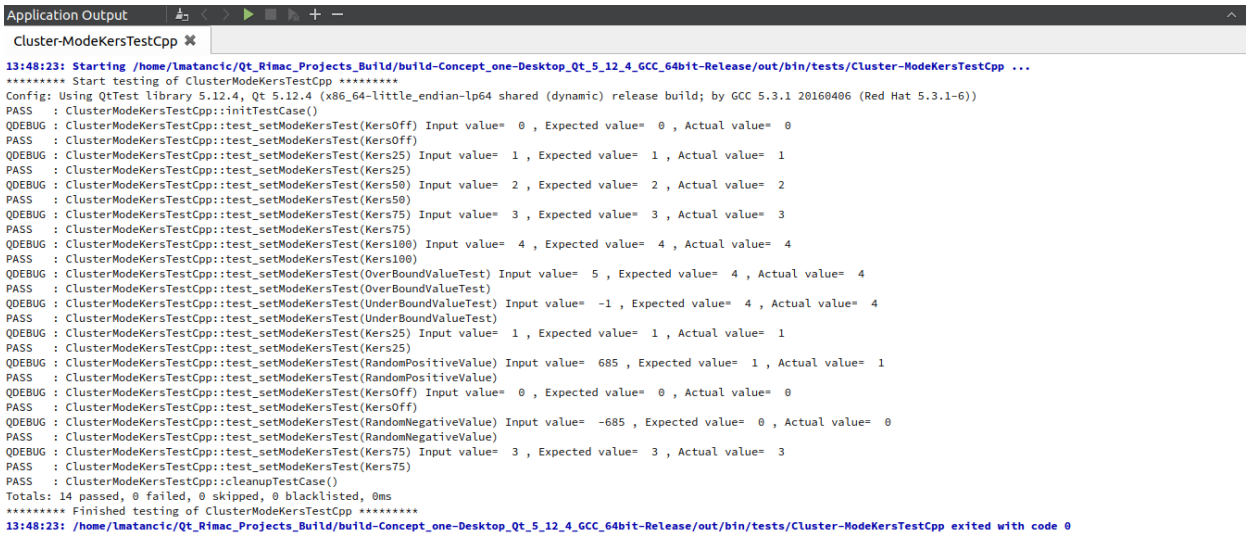
Slika 5.5. *Testiranje centralnog zaslona, QtQuickTest testiranje jedinice, metoda ModeKers*

Prema slici 5.6 prikazan je rezultat testa QML svojstva centralnog zaslona nad metodom ModeGearbox koja predstavlja funkcionalnost podešavanja mjenjača. Nad metodom je provedeno 12 testova, a svi su uspješno prošli testiranje uz vrijeme izvođenja testa od 56 milisekundi.



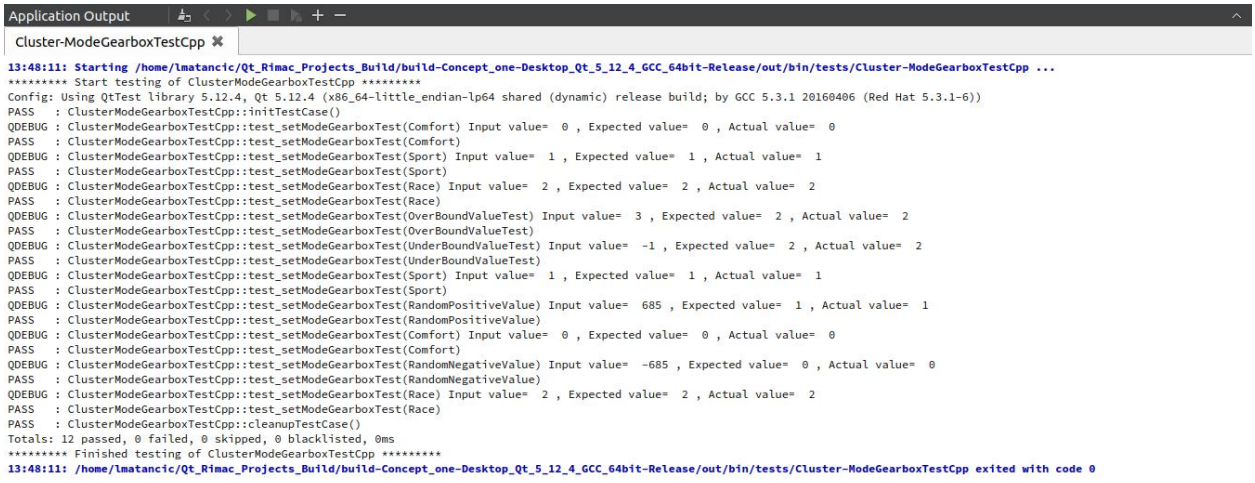
Slika 5.6. *Testiranje centralnog zaslona, QtQuickTest testiranje jedinice, metoda ModeGearbox*

Prema slici 5.7 prikazan je rezultat testiranja jedinice aplikacije digitalne nadzorne ploče koristeći QtTest *framework* nad metodom ModeKers koja ima funkcionalnost podešavanja rekuperacije energije u baterijama koristeći kinetičku energiju kočenja. Izvedeno je 14 testnih slučajeva, a svi su zadovoljili očekivani rezultat uz vrijeme izvođenja testa od 0 milisekundi.



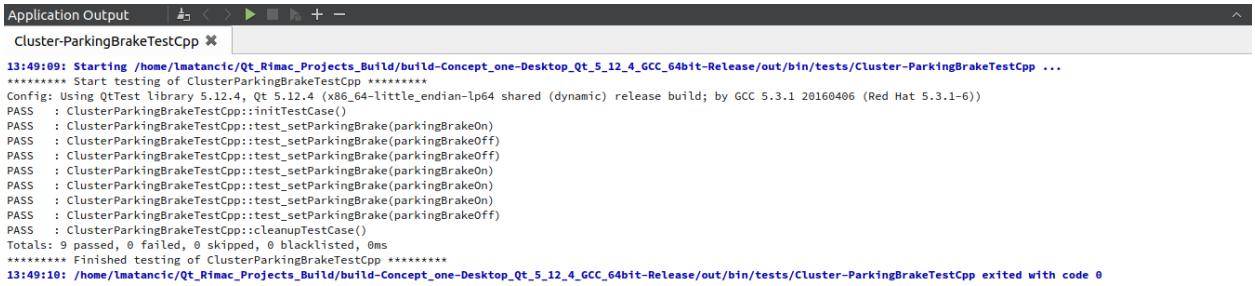
Slika 5.7. *Testiranje digitalne nadzorne ploče jedinice, QtTest test jedinice, metoda ModeKers*

Prema slici 5.8 prikazan je rezultat testiranja izvornog koda C++ jedinice aplikacije digitalne nadzorne ploče pri čemu je nad testiranom metodom ModeGearbox provedeno 12 testova koji su svi uspješno izvedeni u vremenu od 0 milisekundi.



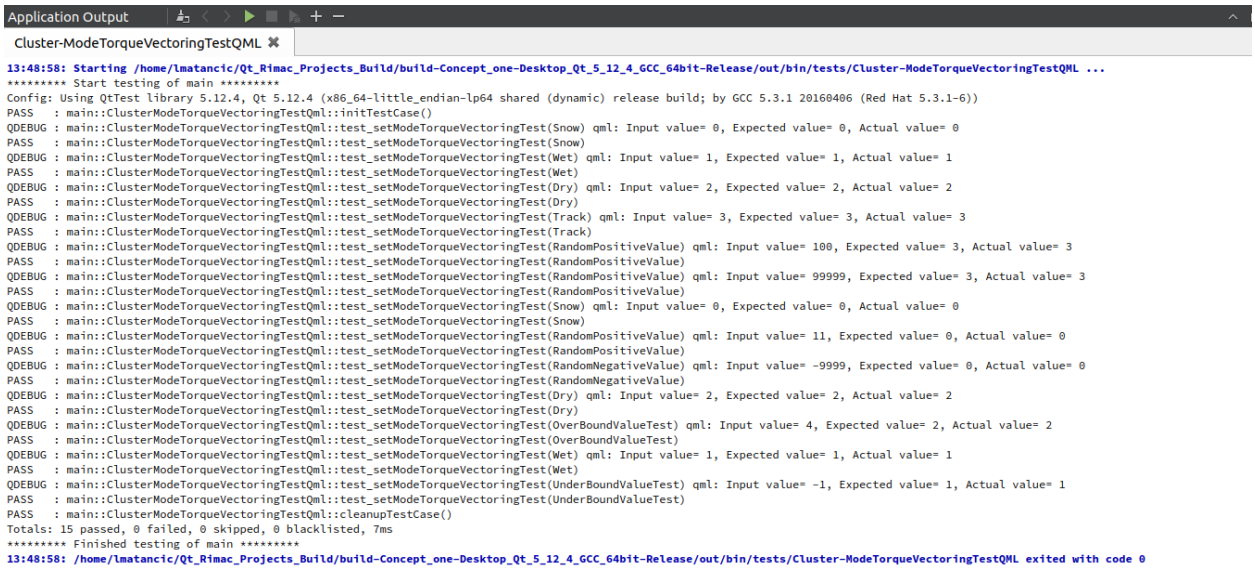
Slika 5.8. *Testiranje digitalne nadzorne ploče, QtTest test jedinice, metoda ModeGearbox*

Prema slici 5.9 prikazan je rezultat testiranja jedinice C++ izvornog koda nad metodom ParkingBrake unutar aplikacije digitalne nadzorne ploče. Metoda je testirana na 9 testnih slučajeva te je u svima uspješno prošla test sa vremenom izvođenja testa od 0 milisekundi.



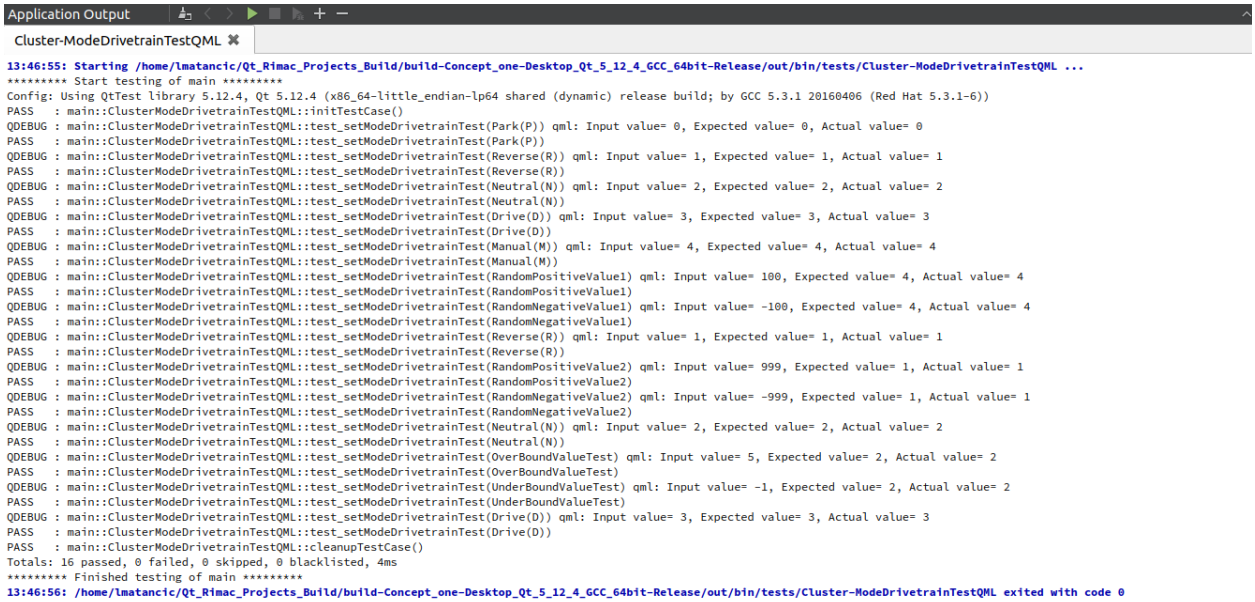
Slika 5.9. *Testiranje digitalne nadzorne ploče, QtTest test jedinice, metoda ParkingBrake*

Rezultat testiranja QML jedinice aplikacije digitalne nadzorne ploče nad metodom ModeTorqueVectoring nad kojom je izvedeno 15 testova i svi su zadovoljili očekivani rezultat za vrijeme od 7 milisekundi prikazan je prema slici 5.10.



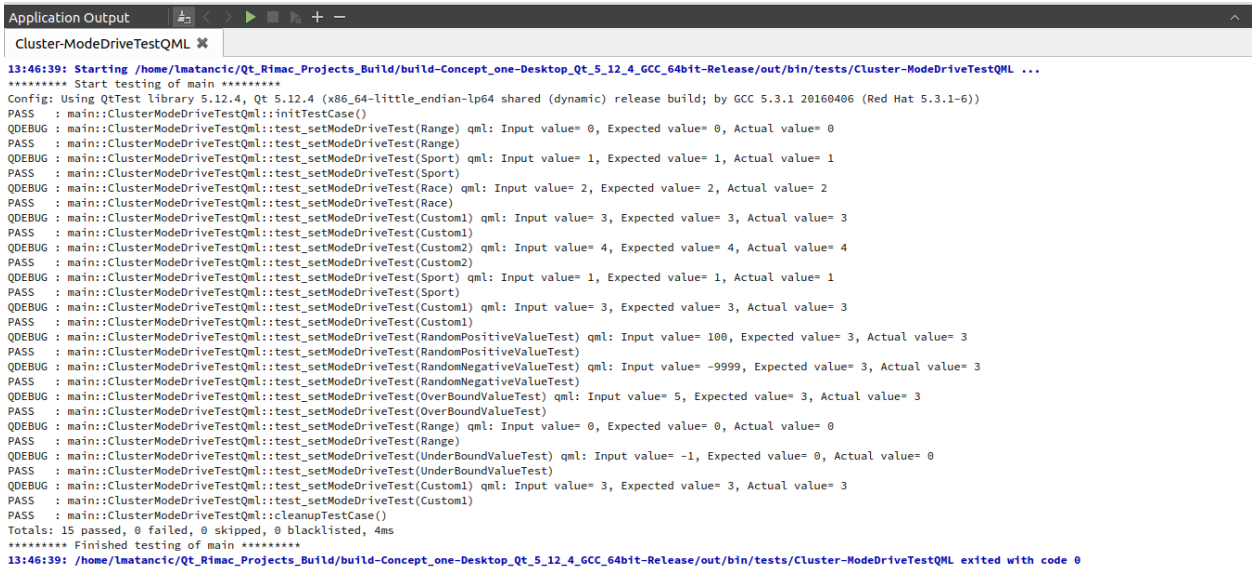
Slika 5.10. *Testiranje digitalne nadzorne ploče, QtQuickTest testiranje jedinice, metoda ModeTorqueVectoring*

Rezultat testiranja QML jedinice aplikacije digitalne nadzorne ploče nad metodom ModeDrivetrain prikazan je prema slici 5.11 Nad metodom je izvedeno 16 testova, a svi su uspješno izvedeni u skladu s očekivanim rezultatom u vremenu od 4 milisekunde.



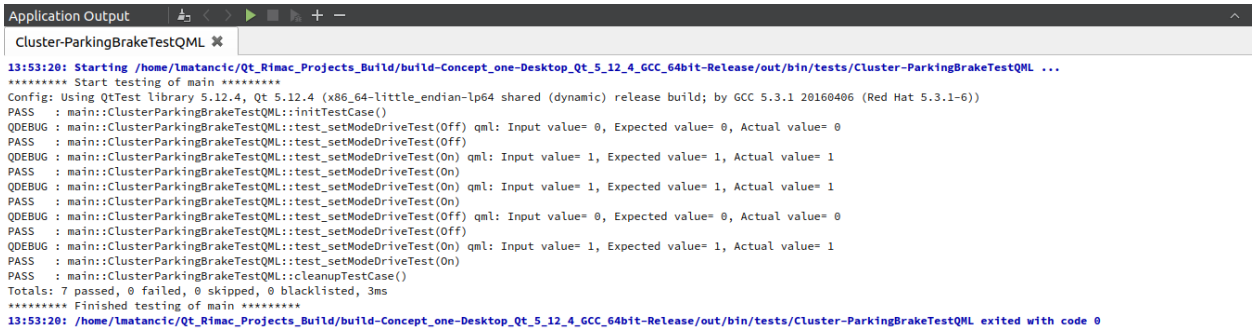
Slika 5.11. *Testiranje digitalne nadzorne ploče, QtQuickTest testiranje jedinice, metoda ModeDrivetrain*

Rezultat testiranja QML jedinice koristeći QtQuickTest nad aplikacijom digitalne nadzorne ploče prikazan je prema slici 5.12 Testirana je metoda ModeDrive nad kojom je provedeno 15 testnih slučajeva koji su uspješno izvedeni i dobiveni su očekivani rezultati, a vrijeme trajanja testova je 4 milisekunde.



Slika 5.12. *Testiranje digitalne nadzorne ploče, QtQuickTest testiranje jedinice, metoda ModeDrive*

Prema slici 5.13 prikazan je rezultat testiranja QML jedinice aplikacije digitalne nadzorne ploče nad metodom Parking Brake. Nad metodom je provedeno 7 testnih slučajeva od kojih su svi uspješno zadovoljili očekivani rezultat unutar vremena od 3 milisekunde.

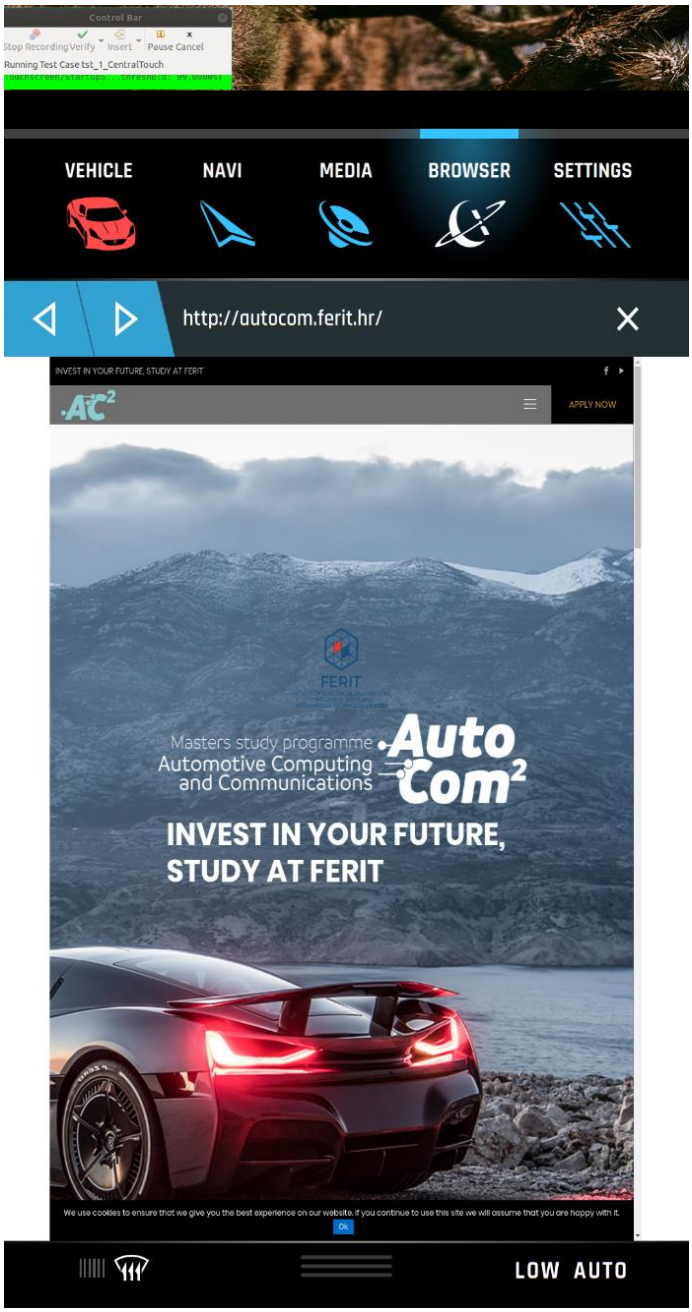


Slika 5.13. *Testiranje digitalne nadzorne ploče, QtQuickTest testiranje jedinice, metoda ParkingBrake*

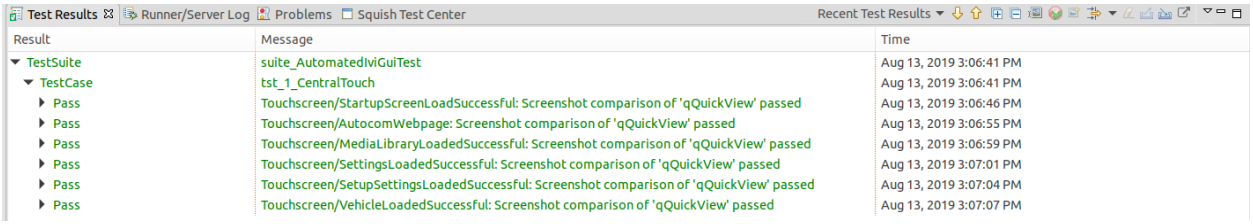
5.3. Rezultati integracijskog testiranja

Kod integracijskog testiranja više se ne vidi samo programski kod sa ispisom rezultata testova već se koriste zasloni unutar Squish alata za testiranje. Pri ovoj vrsti testiranja provjeravaju se odziv aplikacije na ulaz s zaslona i komunikacija između aplikacije koja simulira signale mehaničkih ulaza informacijsko-zabavnog sustava vozila prema centralnom zaslonu ili digitalnoj nadzornoj ploči te se izvodi provjera ispravnosti prikaza na grafičkom korisničkom sučelju određenog zaslona.

Prema slici 5.14 vidljivo je izvođenje testiranja centralnog zaslona pri čemu se testira funkcionalnost zaslona na dodir kod koje se iz Squish alata pokreće aplikacija, te se dodaju unosi na zaslonu, pri čemu se provjerava ispravno učitavanje glavnog izbornika, pomicanje u izbornik sa web tražilicom te se odlazi na stranicu FERIT Autocom[2] pri čemu se izvodi i web test koji provjerava ispravnost elemenata stranice i zaslona na centralnom zaslonu. Nakon toga se izvodi pomicanje u izbornik s glazbom, postavkama, prilagođavanjem postavki te se vraća u glavni izbornik pri čemu se u svakoj točki vrši testiranje snimke zaslona s očekivanom. Nakon izvođenja integracijskog testiranja zaslona na dodir dobiven je rezultat prolazaka svih izvedenih testova prema slici 5.15.



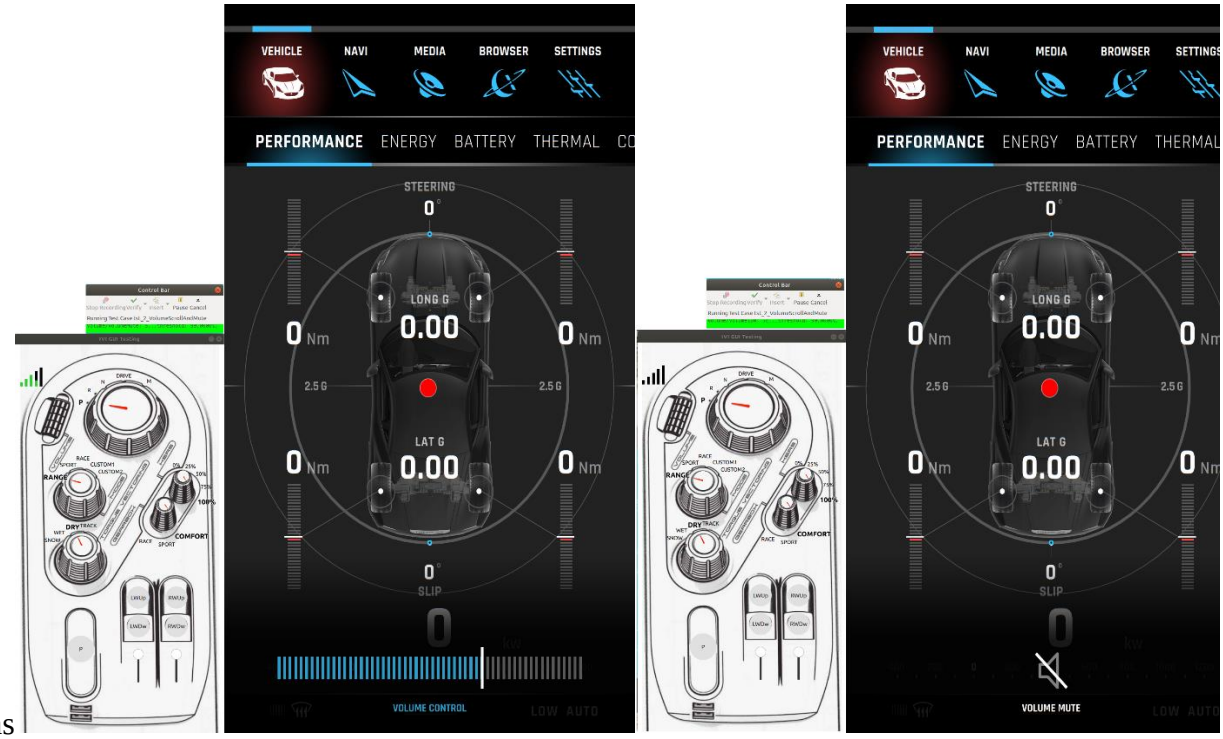
Slika 5.14. *Izvođenje testiranja zaslona na dodir na centralnom zaslonu*



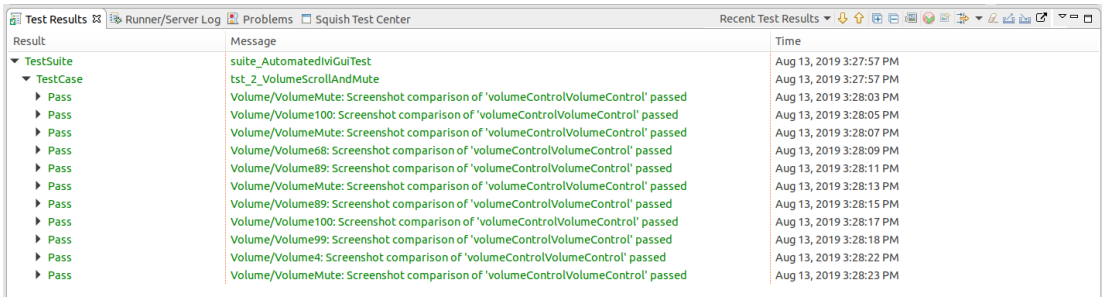
Slika 5.15. *Rezultati testiranja zaslona na dodir na centralnom zaslonu*

Slijedeći test nad aplikacijom centralnog zaslona prikaz je podešavanja je zvuka te gašenja zvuka. Prilikom ovog testa koristi se aplikacija za simulaciju signala mehaničkih ulaza koja komunicira ZMQ komunikacijom sa centralnim zaslonom. Pri izvođenu testa vidljivog na slici 5.16 potrebno je dosta precizno uhvatiti područje zaslona koje se testira i u pravom vremenu pošto se ovaj element pokazuje kao što se prikazuju i ostali elementi centralnog zaslona na njihovu promjenu. Zaslon je testiran na unose pojačanja zvuka, pokušaja pojačanja nakon što je jačina zvuka dosegla gornju granicu, gašenja zvuka, te promjenu stanja jačine zvuka nakon njegovog gašenja koja je omogućena pomicanjem kotačića na stvarnoj izvedbi, odnosno pomicanjem kotačića miša kroz Squish alat na aplikaciji koja simulira signale mehaničkih ulaza. Dobiveni rezultati prema slici 5.17 su u skladu s očekivanima.

as

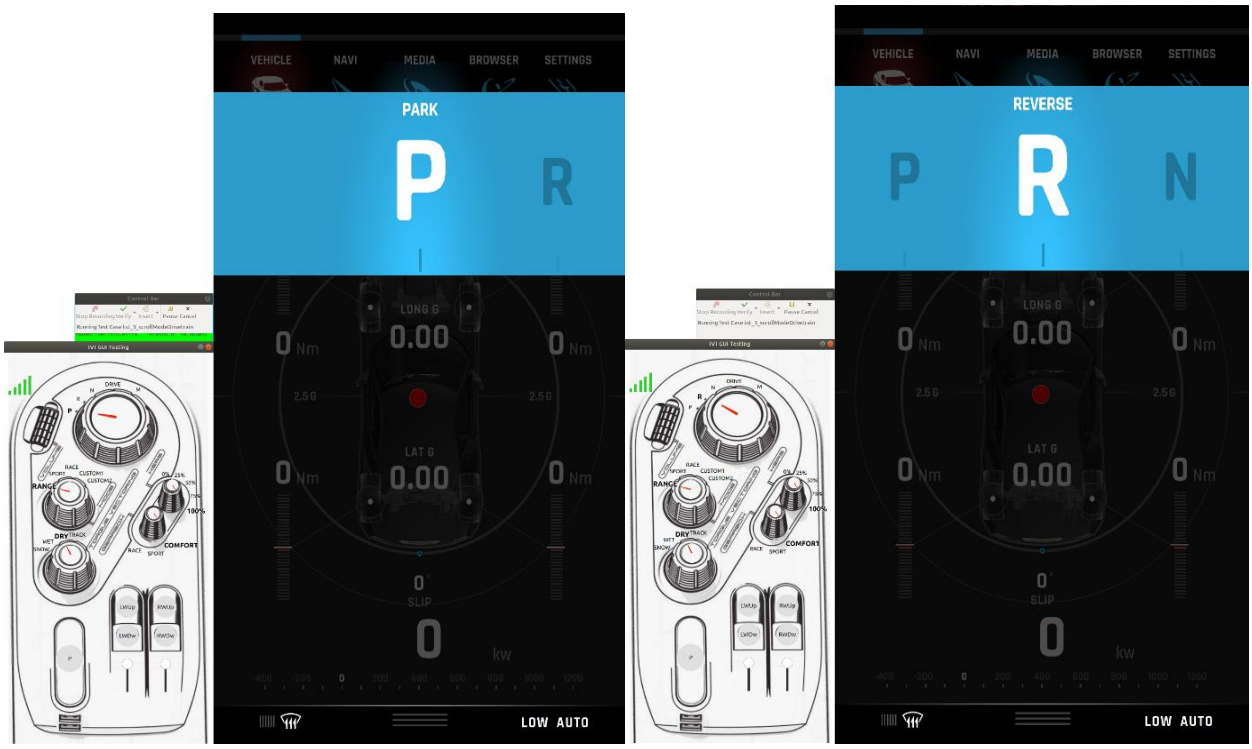


Slika 5.16. *Izvođenje testiranja razine (na slici lijevo), paljenja i gašenja zvuka(na slici desno) na centralnom zaslonu*

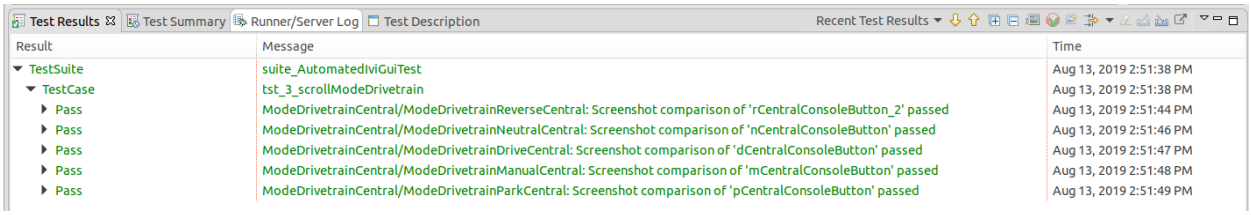


Slika 5.17. *Rezultati testiranja razine, paljenja i gašenja zvuka na centralnom zaslonu*

Prema slici 5.18 prikazano je testiranje prikaza stupnja mjenjača vozila u dva načina od svih mogućih koji su testirani, odnosno prikazano je podešavanje stupnja mjenjača u parking način rada i u način vožnje unatrag, a testirani su još i neutralni način, način vožnje s automatskim i ručnim mjenjačem. Pri tome se također testiranje izvodilo na principu uzimanja slike zaslona u točnom vremenu okidanja animacije uzimajući samo u obzir elemente od interesa unutar verifikacijskog zaslona. Prema slici 5.19 vidljivi su rezultati ovog testa od kojih su svi u skladu s očekivanima.

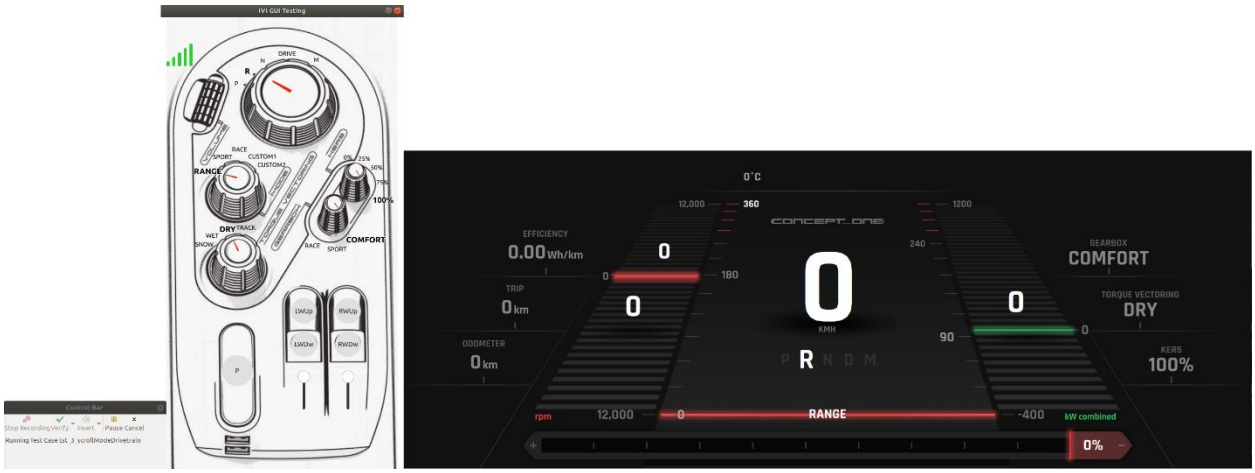


Slika 5.18. *Izvođenje testiranja stupnja mjenjača u parking (lijevo) i unatrag (desno) na centralnom zaslonu*

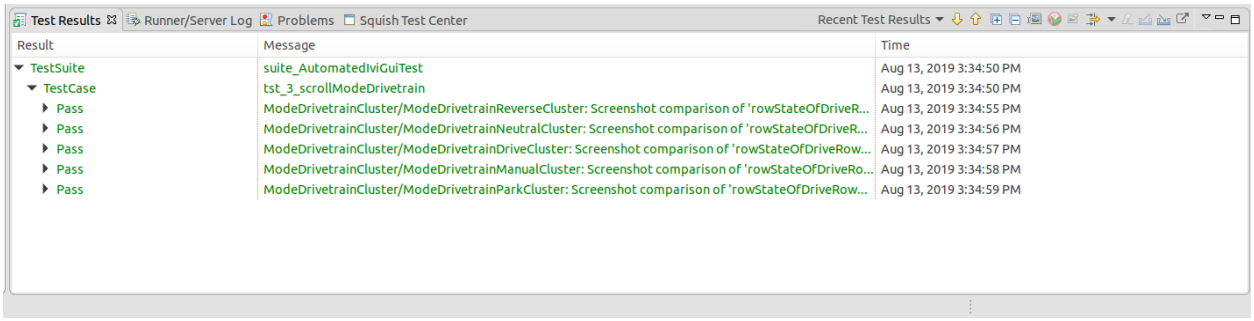


Slika 5.19. *Rezultati testiranja stupnja mjenjača na centralnom zaslonu*

Izvođenje testiranja prikaza stupnja prijenosa na digitalnoj nadzornoj ploči, a pri čemu se ulazi temelje na signalima aplikacije koja simulira mehaničke ulaze vidljivo je na slici 5.20 Na ovom zaslonu elementi koji se prikazuju imaju određeni period prijelazne animacije ali trenutna postavljena vrijednost ostaje prikazana na zaslonu te se prilikom izvođenja testa upravo taj element snimke zaslona traži. Rezultat izvođenja testa prema slici 5.21 prikazuje sukladnost rezultata testiranja sa onima koji su očekivani, odnosno svi testovi su prošli ovaj integracijski test.

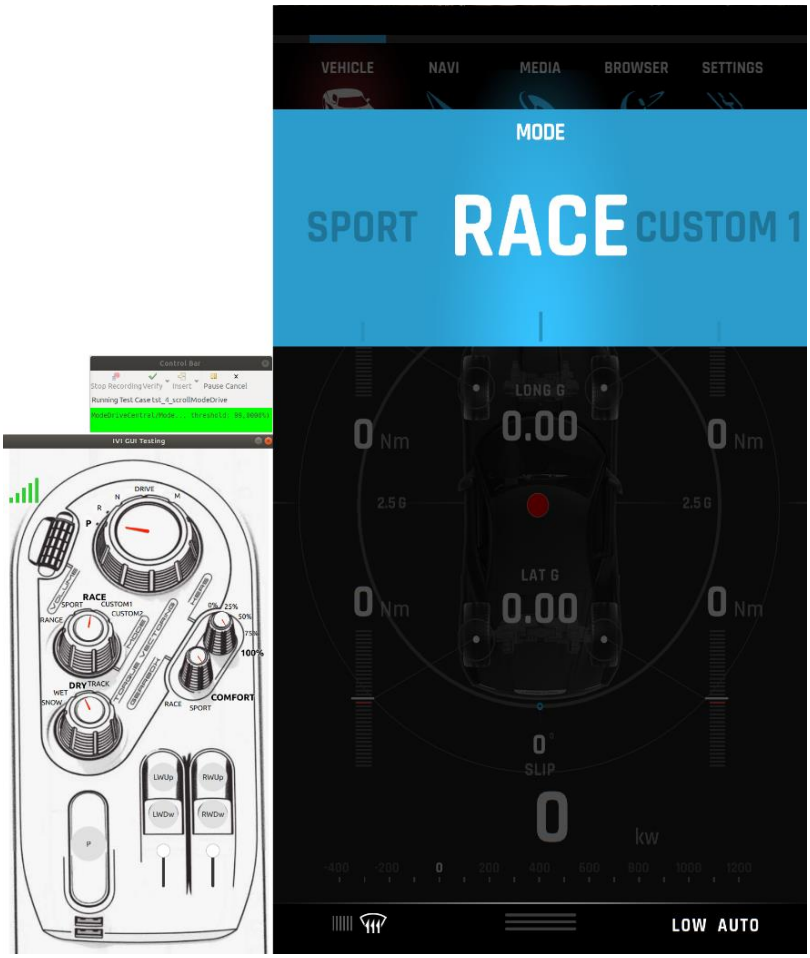


Slika 5.20. *Izvođenje testiranja stupnja prijenosa u unatrag na digitalnoj nadzornoj ploči*

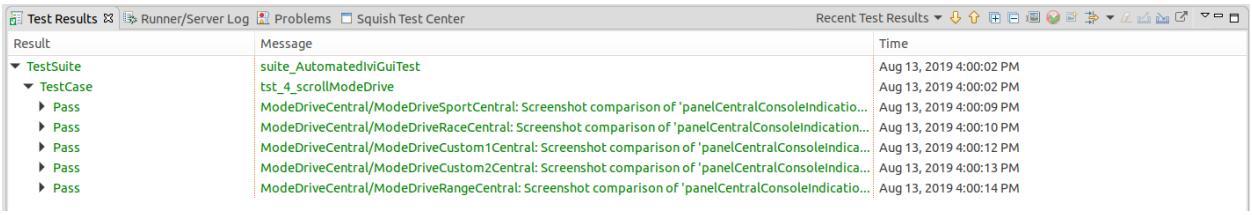


Slika 5.21. *Rezultati testiranja načina pogonskog sustava na digitalnoj nadzornoj ploči*

Izvođenje testiranja načina vožnje korištenjem Squish alata nad aplikacijom centralnog zaslona vidljivo je na slici 5.22 Sukladno prethodno izvedenim testovima, promjena vrijednosti ove funkcionalnosti odrađuje se unutar animacije na centralnom zaslonu. Dobiveni rezultati testiranja su u skladu s očekivanim verifikacijskim točkama testa, a prikazani su na slici 5.23.

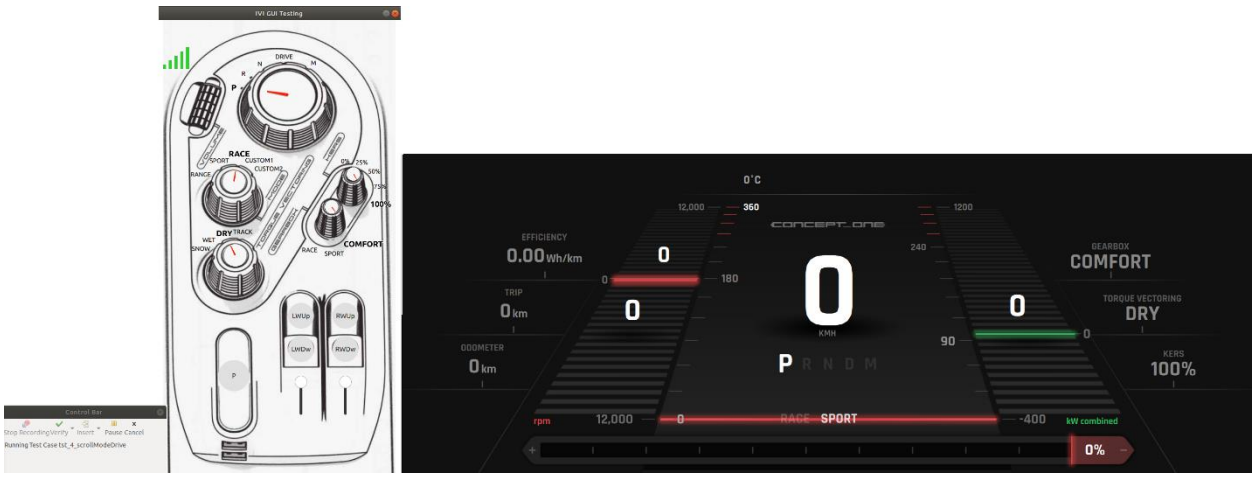


Slika 5.22. *Izvođenje testiranja načina vožnje na centralnom zaslonu*

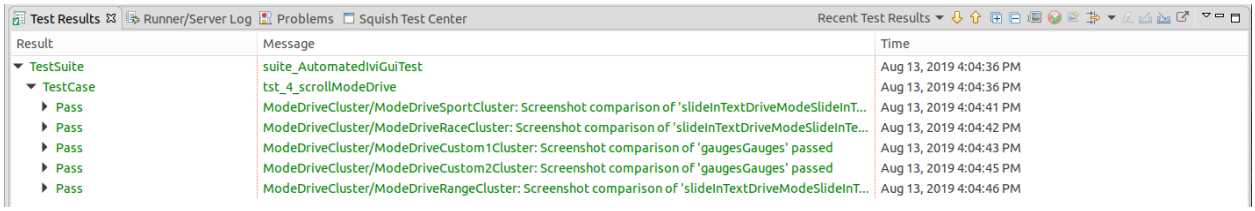


Slika 5.23. *Rezultati testiranja načina vožnje na centralnom zaslonu*

Izvođenje jednog od testova prilikom testiranja načina vožnje na digitalnoj nadzornoj ploči vidljivo je na slici 5.24 Svi testovi unutar ovog integracijskog testa su prošli uspješno, što je vidljivo na slici 5.25.



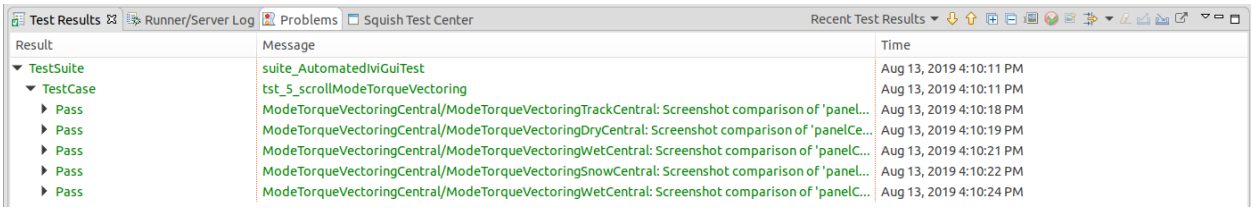
Slika 5.24. *Izvođenje testiranja načina vožnje na digitalnoj nadzornoj ploči*



Slika 5.25. *Rezultati testiranja načina vožnje na digitalnoj nadzornoj ploči*

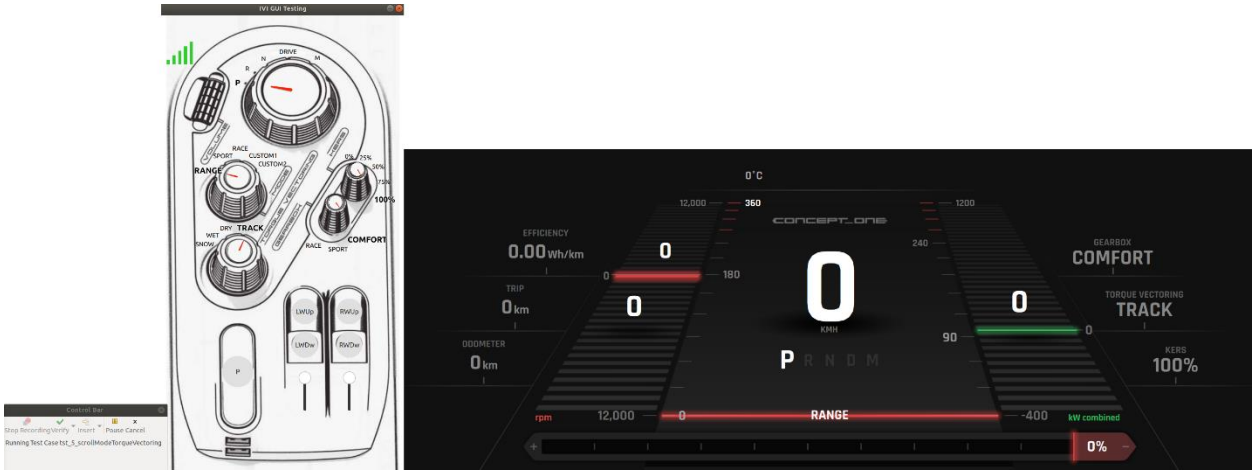
Jedan od testova prikazivanja podešenog načina prijenosa okretnog momenta na kotače vozila na centralnom zaslonu prikazan je na slici 5.26 Testiranje je provedeno za sva postojeća podešenja te su svi testovi uspješno prošli testiranje prema slici 5.27.

Slika 5.26. *Izvođenje testiranja načina prijenosa okretnog momenta na centralnom zaslonu*

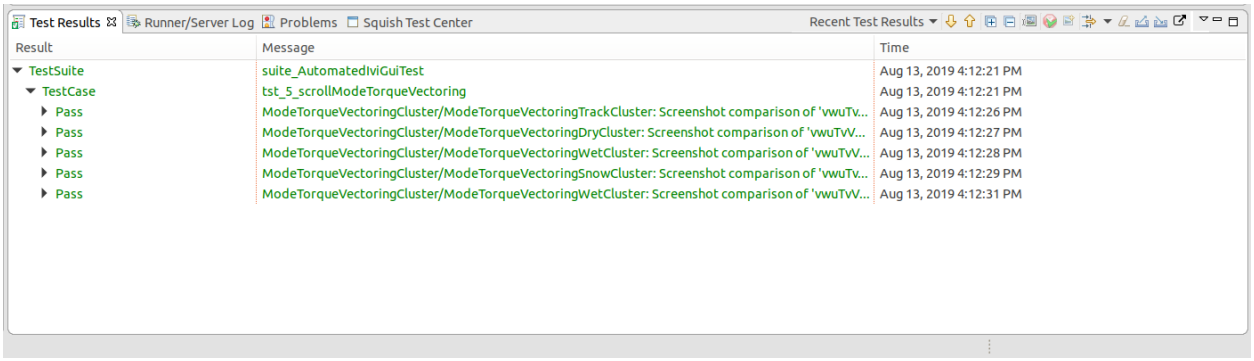


Slika 5.27. *Rezultati testiranja načina prijenosa okretnog momenta na centralnom zaslonu*

Izvođenje testiranja grafičkog sučelja prema slici 5.28 odvija se na ista podešenja kao i na centralnom zaslonu, jedina razlika je što je aplikacija pod testom aplikacija digitalne nadzorne ploče. Dobiveni rezultat prema slici 5.29 prikazuje prolazak svih testova, odnosno poklapanje dobivenih snimaka zaslona na mjestima gdje su odabrani traženi elementi slike.

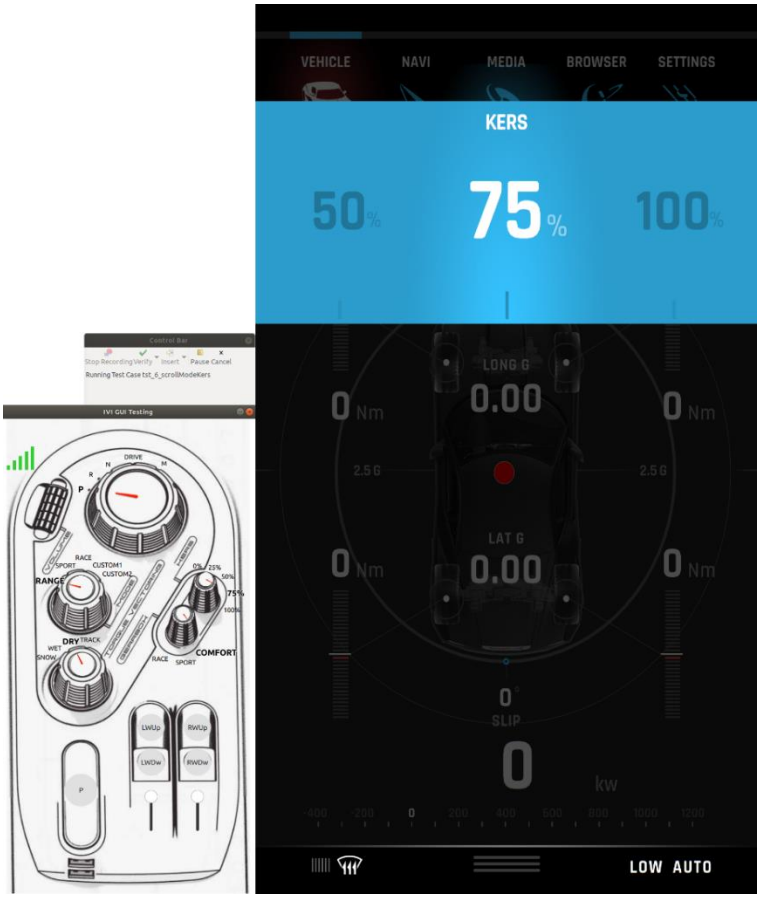


Slika 5.28. *Izvođenje testiranja načina načina prijenosa okretnog momenta na digitalnoj nadzornoj ploči*

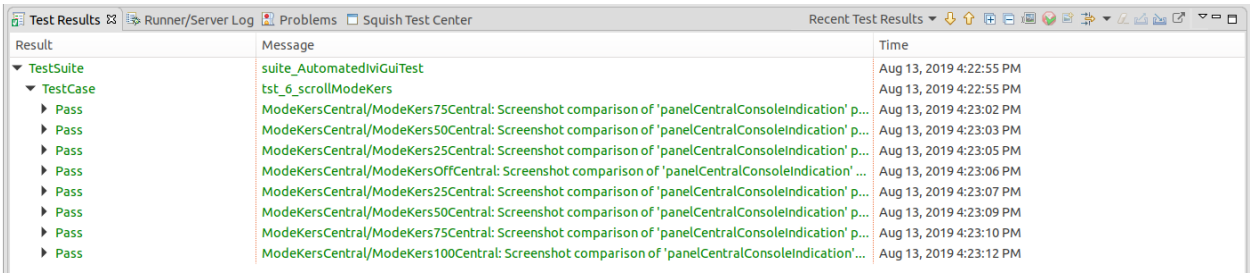


Slika 5.29. *Izvođenje testiranja načina prijenosa okretnog momenta na digitalnoj nadzornoj ploči*

Prema slici 5.30 vidljivo je izvođenje integracijskog testa grafičkog korisničko sučelja aplikacije centralnog zaslona na podešavanje rekuperacije energije baterije preko kinetičke energije kočenja. Rezultati testova prema slici 5.31 da je prolazak svih verifikacijskih točaka testa uspješan.

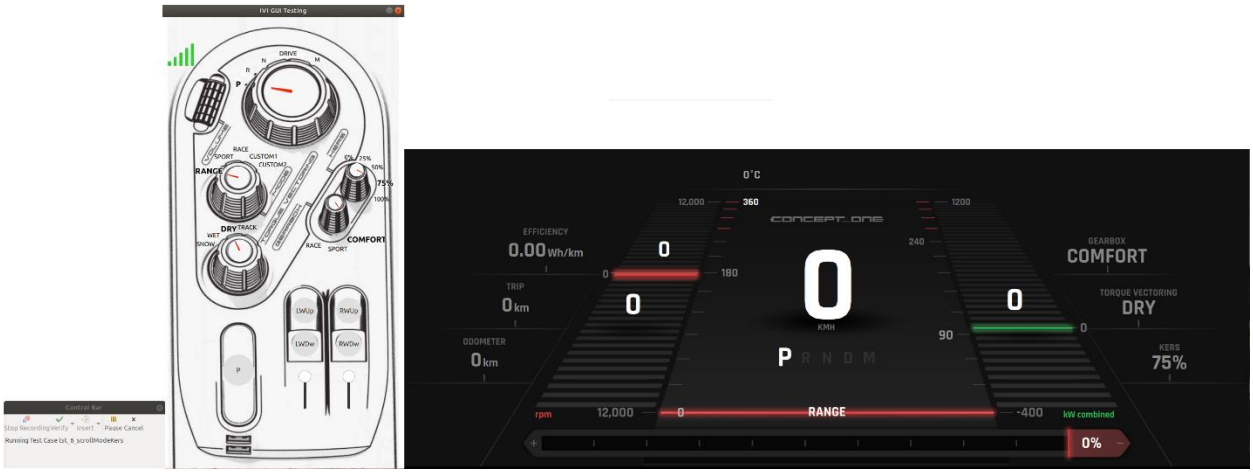


Slika 5.30. *Izvođenje testiranja podešenja rekuperacije energije na centralnom zaslonu*

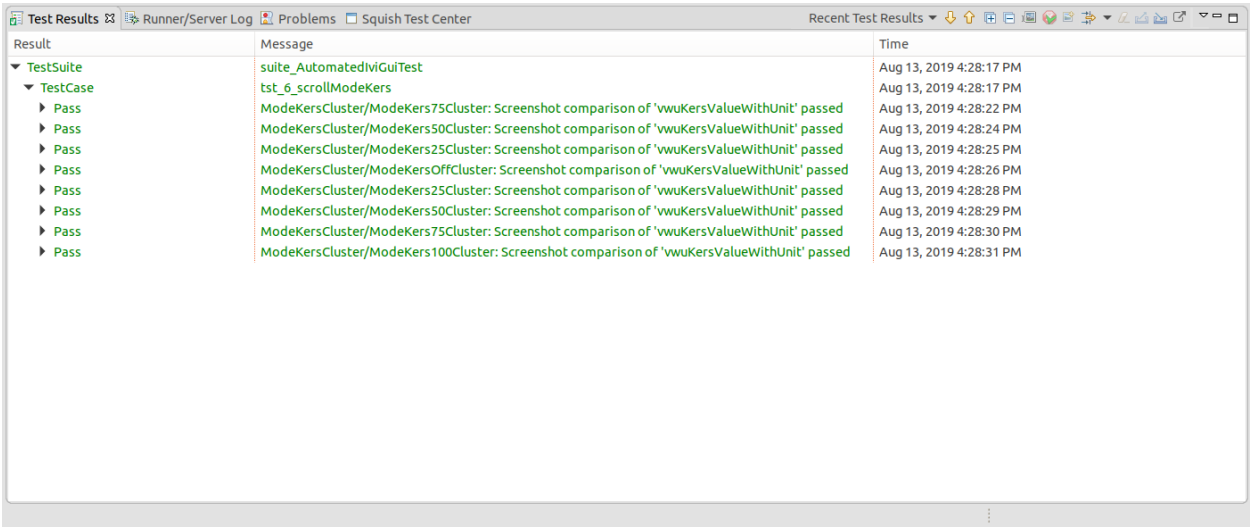


Slika 5.31. *Rezultati testiranja podešenja rekuperacije energije na centralnom zaslonu*

Izvođenje integracijskog testa grafičkog korisničkog sučelja na funkcionalnost podešavanja rekuperacije energije na digitalnoj nadzornoj ploči prikazano je na slici 5.32. Signali koji su poslani iz aplikacije koja simulira mehaničke ulaze isti su kao i kod centralnog zaslona. Dobiveni rezultati testiranjem ove funkcionalnosti na digitalnoj nadzornoj ploči prema slici 5.33 prikazuju da su svi testovi uspješno izvedeni.



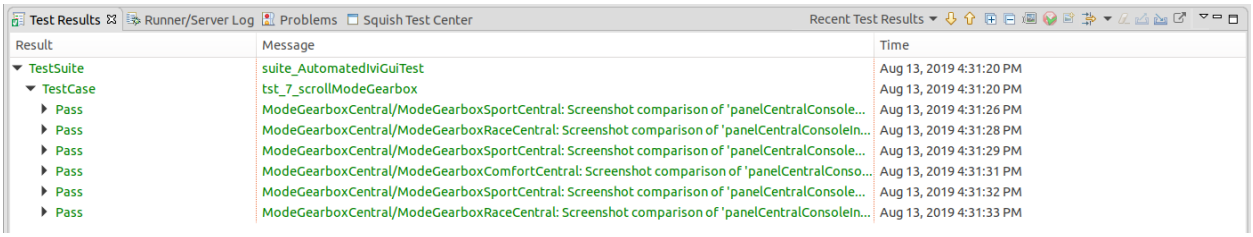
Slika 5.32. *Izvođenje testiranja podešenja rekuperacije energije na digitalnoj nadzornoj ploči*



Slika 5.33. *Izvođenje testiranja podešenja rekuperacije energije na digitalnoj nadzornoj ploči*

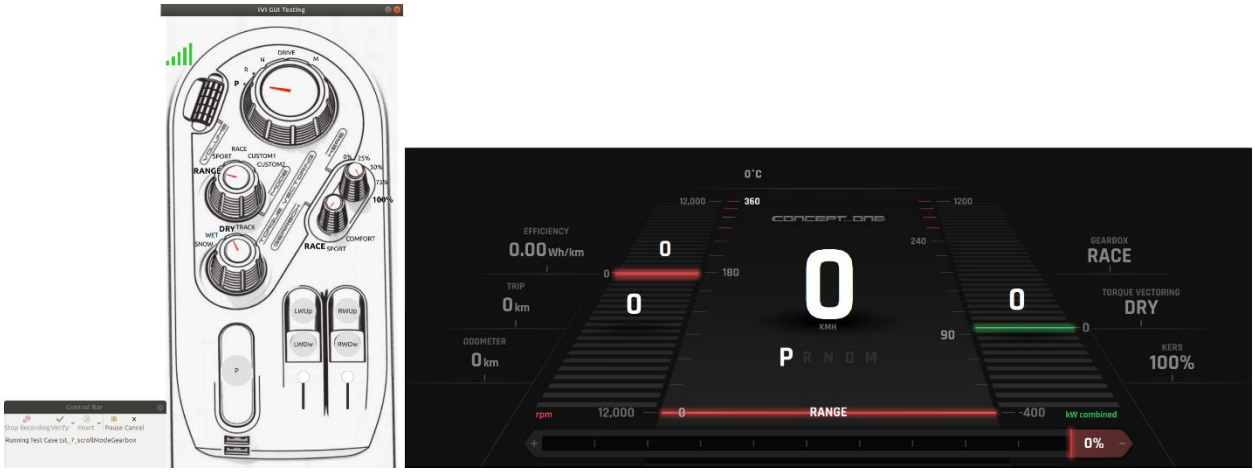
Testiranje prikaza podešavanja mjenjača na grafičkom korisničkom sučelju aplikacije centralnog zaslona vidljivo je na slici 5.34. Rezultati koji su dobiveni testiranjem prema slici 5.35 prikazuju uspješan prolazak svih zadanih testova kroz verifikacijske točke.

Slika 5.34. *Izvođenje testiranja podešavanja rada mjenjača na centralnom zaslonu*

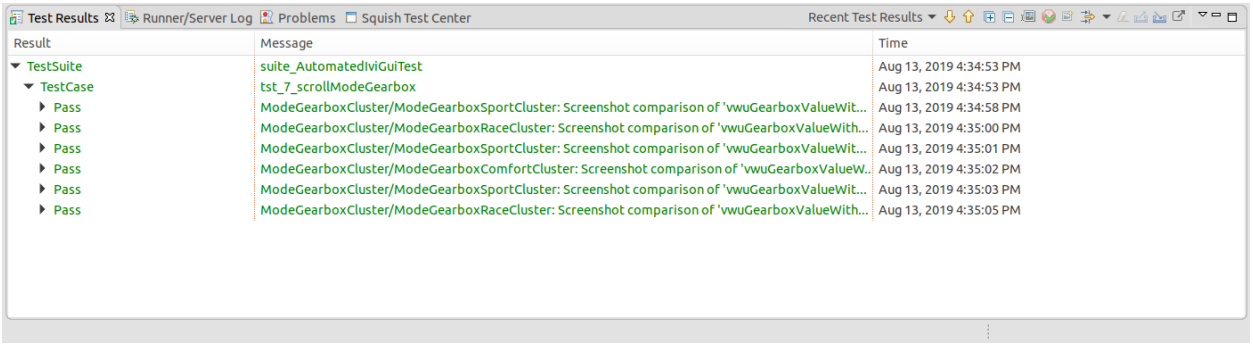


Slika 5.35. *Rezultati testiranja podešavanja rada mjenjača na centralnom zaslonu*

Testiranje prikaza podešavanja rada mjenjača na grafičkom korisničkom sučelju aplikacije digitalne nadzorne ploče prikazano je na slici 5.36 Signali koje šalje aplikacija koja simulira mehaničke ulaze aplikacije identični su onima kao i kod testiranja aplikacije centralnog zaslona na ovu funkcionalnost. Dobiveni rezultati prema slici 5.37 prikazuju uspješnost prolaska svih testova.

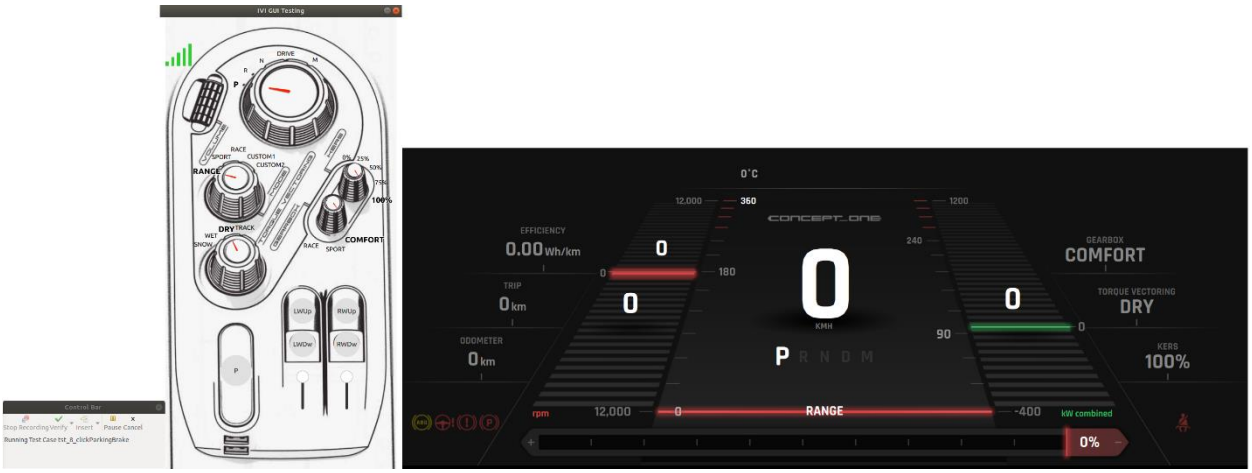


Slika 5.36. *Izvođenje testiranja podešavanja rada mjenjača na digitalnoj nadzornoj ploči*



Slika 5.37. *Rezultati testiranja podešavanja rada mjenjača na digitalnoj nadzornoj ploči*

Izvođenje testiranja prikaza stanja elektroničke parkirne kočnice na grafičkom korisničkom sučelju aplikacije digitalne nadzorne ploče vidljivo je prema slici 5.38 Rezultati izvođenja ovog testiranja vidljivi su prema slici 5.39, a vidljivo je kako su svi testovi uspješno prošli verifikacijske točke.



Slika 5.38. *Izvođenje testiranja stanja parkirne kočnice na digitalnoj nadzornoj ploči*

Slika 5.39. *Rezultati testiranja stanja parkirne kočnice na digitalnoj nadzornoj ploči*

5.4. Rezultati testiranja sustava

Prikaz funkcionalnih i nefunkcionalih rezultata automatiziranog sustavskog testiranja vidljivi su prema slici 5.40. Iz slike rezultata vidljivo je kako su svi testovi uspješno prošli verifikacijske točke te kako se mijenja uporaba memorije svake aplikacije kroz postupak izvođenja testa.

Test Results | Test Summary | Runner/Server Log | Test Description — Recent Test Results

Result	Message	Time
TestSuite	suite_AutomatedIviGuiTest	Aug 13, 2019 9:34:30 AM
TestCase	tst_SystemTesting	Aug 13, 2019 9:34:30 AM
Log	Executing: tst_1_CentralTouch	Aug 13, 2019 9:34:30 AM
Log	Memory usage: 265.6640625MB	Aug 13, 2019 9:34:35 AM
Log	Process ID: 20078	Aug 13, 2019 9:34:35 AM
Log	Memory usage: 158.7421875MB	Aug 13, 2019 9:34:35 AM
Log	Process ID: 20155	Aug 13, 2019 9:34:35 AM
Log	Memory usage: 95.0234375MB	Aug 13, 2019 9:34:35 AM
Log	Process ID: 20209	Aug 13, 2019 9:34:35 AM
Pass	Touchscreen/StartupScreenLoadSuccessful: Screenshot comparison of 'qQuickView' passed	Aug 13, 2019 9:34:37 AM
Pass	Touchscreen/AutocomWebpage: Screenshot comparison of 'qQuickView' passed	Aug 13, 2019 9:34:46 AM
Pass	Touchscreen/MediaLibraryLoadedSuccessful: Screenshot comparison of 'qQuickView' passed	Aug 13, 2019 9:34:51 AM
Pass	Touchscreen/SettingsLoadedSuccessful: Screenshot comparison of 'qQuickView' passed	Aug 13, 2019 9:34:55 AM
Pass	Touchscreen/SetupSettingsLoadedSuccessful: Screenshot comparison of 'qQuickView' passed	Aug 13, 2019 9:34:58 AM
Pass	Touchscreen/VehicleLoadedSuccessful: Screenshot comparison of 'qQuickView' passed	Aug 13, 2019 9:35:02 AM
Log	Memory usage: 377.54296875MB	Aug 13, 2019 9:35:04 AM
Log	Process ID: 20078	Aug 13, 2019 9:35:04 AM
Log	Memory usage: 158.796875MB	Aug 13, 2019 9:35:04 AM
Log	Process ID: 20155	Aug 13, 2019 9:35:04 AM
Log	Memory usage: 95.0234375MB	Aug 13, 2019 9:35:04 AM
Log	Process ID: 20209	Aug 13, 2019 9:35:04 AM
Log	Executing: tst_2_VolumeScrollAndMute	Aug 13, 2019 9:35:05 AM
Pass	Volume/VolumeMute: Screenshot comparison of 'volumeControlVolumeControl' passed	Aug 13, 2019 9:35:07 AM
Pass	Volume/Volume100: Screenshot comparison of 'volumeControlVolumeControl' passed	Aug 13, 2019 9:35:09 AM
Pass	Volume/VolumeMute: Screenshot comparison of 'volumeControlVolumeControl' passed	Aug 13, 2019 9:35:11 AM
Pass	Volume/Volume68: Screenshot comparison of 'volumeControlVolumeControl' passed	Aug 13, 2019 9:35:13 AM
Pass	Volume/Volume89: Screenshot comparison of 'volumeControlVolumeControl' passed	Aug 13, 2019 9:35:15 AM
Pass	Volume/VolumeMute: Screenshot comparison of 'volumeControlVolumeControl' passed	Aug 13, 2019 9:35:17 AM
Pass	Volume/Volume89: Screenshot comparison of 'volumeControlVolumeControl' passed	Aug 13, 2019 9:35:19 AM
Pass	Volume/Volume100: Screenshot comparison of 'volumeControlVolumeControl' passed	Aug 13, 2019 9:35:21 AM
Pass	Volume/Volume99: Screenshot comparison of 'volumeControlVolumeControl' passed	Aug 13, 2019 9:35:23 AM
Pass	Volume/Volume4: Screenshot comparison of 'volumeControlVolumeControl' passed	Aug 13, 2019 9:35:26 AM
Pass	Volume/VolumeMute: Screenshot comparison of 'volumeControlVolumeControl' passed	Aug 13, 2019 9:35:28 AM
Log	Memory usage: 377.64453125MB	Aug 13, 2019 9:35:29 AM
Log	Process ID: 20078	Aug 13, 2019 9:35:29 AM
Log	Memory usage: 158.81640625MB	Aug 13, 2019 9:35:29 AM
Log	Process ID: 20155	Aug 13, 2019 9:35:29 AM
Log	Memory usage: 112.21875MB	Aug 13, 2019 9:35:29 AM
Log	Process ID: 20209	Aug 13, 2019 9:35:29 AM
Log	Executing: tst_3_scrollModeDrivetrain	Aug 13, 2019 9:35:30 AM
Pass	ModeDrivetrainCentral/ModeDrivetrainReverseCentral: Screenshot comparison of 'rCentralConsoleButton_2' passed	Aug 13, 2019 9:35:31 AM
Pass	ModeDrivetrainCluster/ModeDrivetrainReverseCluster: Screenshot comparison of 'rowStateOfDriveRow' passed	Aug 13, 2019 9:35:33 AM
Pass	ModeDrivetrainCentral/ModeDrivetrainNeutralCentral: Screenshot comparison of 'nCentralConsoleButton' passed	Aug 13, 2019 9:35:34 AM
Pass	ModeDrivetrainCluster/ModeDrivetrainNeutralCluster: Screenshot comparison of 'rowStateOfDriveRow' passed	Aug 13, 2019 9:35:35 AM
Pass	ModeDrivetrainCentral/ModeDrivetrainDriveCentral: Screenshot comparison of 'dCentralConsoleButton' passed	Aug 13, 2019 9:35:36 AM
Pass	ModeDrivetrainCluster/ModeDrivetrainDriveCluster: Screenshot comparison of 'rowStateOfDriveRow' passed	Aug 13, 2019 9:35:37 AM
Pass	ModeDrivetrainCentral/ModeDrivetrainManualCentral: Screenshot comparison of 'mCentralConsoleButton' passed	Aug 13, 2019 9:35:39 AM
Pass	ModeDrivetrainCluster/ModeDrivetrainManualCluster: Screenshot comparison of 'rowStateOfDriveRow' passed	Aug 13, 2019 9:35:40 AM
Pass	ModeDrivetrainCentral/ModeDrivetrainParkCentral: Screenshot comparison of 'pCentralConsoleButton' passed	Aug 13, 2019 9:35:41 AM
Pass	ModeDrivetrainCluster/ModeDrivetrainParkCluster: Screenshot comparison of 'rowStateOfDriveRow' passed	Aug 13, 2019 9:35:42 AM
Log	Memory usage: 377.77734375MB	Aug 13, 2019 9:35:43 AM
Log	Process ID: 20078	Aug 13, 2019 9:35:43 AM
Log	Memory usage: 159.07421875MB	Aug 13, 2019 9:35:43 AM
Log	Process ID: 20155	Aug 13, 2019 9:35:43 AM
Log	Memory usage: 120.01953125MB	Aug 13, 2019 9:35:43 AM
Log	Process ID: 20209	Aug 13, 2019 9:35:43 AM
Log	Executing: tst_4_scrollModeDrive	Aug 13, 2019 9:35:44 AM
Pass	ModeDriveCentral/ModeDriveSportCentral: Screenshot comparison of 'panelCentralConsoleIndication' passed	Aug 13, 2019 9:35:46 AM
Pass	ModeDriveCluster/ModeDriveSportCluster: Screenshot comparison of 'slideInTextDriveModeSlideInText' passed	Aug 13, 2019 9:35:47 AM
Pass	ModeDriveCentral/ModeDriveRaceCentral: Screenshot comparison of 'panelCentralConsoleIndication' passed	Aug 13, 2019 9:35:48 AM
Pass	ModeDriveCluster/ModeDriveRaceCluster: Screenshot comparison of 'slideInTextDriveModeSlideInText' passed	Aug 13, 2019 9:35:50 AM
Pass	ModeDriveCentral/ModeDriveCustom1Central: Screenshot comparison of 'panelCentralConsoleIndication' passed	Aug 13, 2019 9:35:51 AM
Pass	ModeDriveCluster/ModeDriveCustom1Cluster: Screenshot comparison of 'gaugesGauges' passed	Aug 13, 2019 9:35:52 AM
Pass	ModeDriveCentral/ModeDriveCustom2Central: Screenshot comparison of 'panelCentralConsoleIndication' passed	Aug 13, 2019 9:35:54 AM
Pass	ModeDriveCluster/ModeDriveCustom2Cluster: Screenshot comparison of 'gaugesGauges' passed	Aug 13, 2019 9:35:55 AM
Pass	ModeDriveCentral/ModeDriveRangeCentral: Screenshot comparison of 'panelCentralConsoleIndication' passed	Aug 13, 2019 9:35:57 AM
Pass	ModeDriveCluster/ModeDriveRangeCluster: Screenshot comparison of 'slideInTextDriveModeSlideInText' passed	Aug 13, 2019 9:35:58 AM
Pass	ModeDriveCluster/ModeDriveRangeCluster: Screenshot comparison of 'slideInTextDriveModeSlideInText' passed	Aug 13, 2019 9:35:58 AM
Log	Memory usage: 378.046875MB	Aug 13, 2019 9:35:59 AM
Log	Process ID: 20078	Aug 13, 2019 9:35:59 AM
Log	Memory usage: 169.7265625MB	Aug 13, 2019 9:35:59 AM
Log	Process ID: 20155	Aug 13, 2019 9:35:59 AM
Log	Memory usage: 120.1015625MB	Aug 13, 2019 9:35:59 AM
Log	Process ID: 20209	Aug 13, 2019 9:35:59 AM

Log	Memory usage: 378.046875MB	Aug 13, 2019 9:35:59 AM
Log	Process ID: 20078	Aug 13, 2019 9:35:59 AM
Log	Memory usage: 169.7265625MB	Aug 13, 2019 9:35:59 AM
Log	Process ID: 20155	Aug 13, 2019 9:35:59 AM
Log	Memory usage: 120.1015625MB	Aug 13, 2019 9:35:59 AM
Log	Process ID: 20209	Aug 13, 2019 9:35:59 AM
Log	Executing: tst_5_scrollModeTorqueVectoring	Aug 13, 2019 9:36:00 AM
Pass	ModeTorqueVectoringCentral/ModeTorqueVectoringTrackCentral: Screenshot comparison of 'panelCentralConsoleIndication' pass...	Aug 13, 2019 9:36:01 AM
Pass	ModeTorqueVectoringCluster/ModeTorqueVectoringTrackCluster: Screenshot comparison of 'vwuTvValueWithUnit' passed	Aug 13, 2019 9:36:02 AM
Pass	ModeTorqueVectoringCentral/ModeTorqueVectoringDryCentral: Screenshot comparison of 'panelCentralConsoleIndication' passed...	Aug 13, 2019 9:36:03 AM
Pass	ModeTorqueVectoringCluster/ModeTorqueVectoringDryCluster: Screenshot comparison of 'vwuTvValueWithUnit' passed	Aug 13, 2019 9:36:05 AM
Pass	ModeTorqueVectoringCentral/ModeTorqueVectoringWetCentral: Screenshot comparison of 'panelCentralConsoleIndication' passe...	Aug 13, 2019 9:36:06 AM
Pass	ModeTorqueVectoringCluster/ModeTorqueVectoringWetCluster: Screenshot comparison of 'vwuTvValueWithUnit' passed	Aug 13, 2019 9:36:07 AM
Pass	ModeTorqueVectoringCentral/ModeTorqueVectoringSnowCentral: Screenshot comparison of 'panelCentralConsoleIndication' pass...	Aug 13, 2019 9:36:09 AM
Pass	ModeTorqueVectoringCluster/ModeTorqueVectoringSnowCluster: Screenshot comparison of 'vwuTvValueWithUnit' passed	Aug 13, 2019 9:36:10 AM
Pass	ModeTorqueVectoringCentral/ModeTorqueVectoringWetCentral: Screenshot comparison of 'panelCentralConsoleIndication' passe...	Aug 13, 2019 9:36:11 AM
Pass	ModeTorqueVectoringCluster/ModeTorqueVectoringWetCluster: Screenshot comparison of 'vwuTvValueWithUnit' passed	Aug 13, 2019 9:36:12 AM
Log	Memory usage: 378.15625MB	Aug 13, 2019 9:36:13 AM
Log	Process ID: 20078	Aug 13, 2019 9:36:13 AM
Log	Memory usage: 169.73828125MB	Aug 13, 2019 9:36:13 AM
Log	Process ID: 20155	Aug 13, 2019 9:36:13 AM
Log	Memory usage: 120.1640625MB	Aug 13, 2019 9:36:13 AM
Log	Process ID: 20209	Aug 13, 2019 9:36:13 AM
Log	Executing: tst_6_scrollModeKers	Aug 13, 2019 9:36:14 AM
Pass	ModeKersCentral/ModeKers75Central: Screenshot comparison of 'panelCentralConsoleIndication' passed	Aug 13, 2019 9:36:16 AM
Pass	ModeKersCluster/ModeKers75Cluster: Screenshot comparison of 'vwuKersValueWithUnit' passed	Aug 13, 2019 9:36:17 AM
Pass	ModeKersCentral/ModeKers50Central: Screenshot comparison of 'panelCentralConsoleIndication' passed	Aug 13, 2019 9:36:19 AM
Pass	ModeKersCluster/ModeKers50Cluster: Screenshot comparison of 'vwuKersValueWithUnit' passed	Aug 13, 2019 9:36:20 AM
Pass	ModeKersCentral/ModeKers25Central: Screenshot comparison of 'panelCentralConsoleIndication' passed	Aug 13, 2019 9:36:21 AM
Pass	ModeKersCluster/ModeKers25Cluster: Screenshot comparison of 'vwuKersValueWithUnit' passed	Aug 13, 2019 9:36:23 AM
Pass	ModeKersCentral/ModeKersOffCentral: Screenshot comparison of 'panelCentralConsoleIndication' passed	Aug 13, 2019 9:36:24 AM
Pass	ModeKersCluster/ModeKersOffCluster: Screenshot comparison of 'vwuKersValueWithUnit' passed	Aug 13, 2019 9:36:25 AM
Pass	ModeKersCentral/ModeKers25Central: Screenshot comparison of 'panelCentralConsoleIndication' passed	Aug 13, 2019 9:36:26 AM
Pass	ModeKersCluster/ModeKers25Cluster: Screenshot comparison of 'vwuKersValueWithUnit' passed	Aug 13, 2019 9:36:28 AM
Pass	ModeKersCentral/ModeKers50Central: Screenshot comparison of 'panelCentralConsoleIndication' passed	Aug 13, 2019 9:36:29 AM
Pass	ModeKersCluster/ModeKers50Cluster: Screenshot comparison of 'vwuKersValueWithUnit' passed	Aug 13, 2019 9:36:30 AM
Pass	ModeKersCentral/ModeKers75Central: Screenshot comparison of 'panelCentralConsoleIndication' passed	Aug 13, 2019 9:36:32 AM
Pass	ModeKersCluster/ModeKers75Cluster: Screenshot comparison of 'vwuKersValueWithUnit' passed	Aug 13, 2019 9:36:33 AM
Pass	ModeKersCentral/ModeKers100Central: Screenshot comparison of 'panelCentralConsoleIndication' passed	Aug 13, 2019 9:36:34 AM
Pass	ModeKersCluster/ModeKers100Cluster: Screenshot comparison of 'vwuKersValueWithUnit' passed	Aug 13, 2019 9:36:35 AM
Log	Memory usage: 378.25MB	Aug 13, 2019 9:36:37 AM
Log	Process ID: 20078	Aug 13, 2019 9:36:37 AM
Log	Memory usage: 169.7578125MB	Aug 13, 2019 9:36:37 AM
Log	Process ID: 20155	Aug 13, 2019 9:36:37 AM
Log	Memory usage: 120.26171875MB	Aug 13, 2019 9:36:37 AM
Log	Process ID: 20209	Aug 13, 2019 9:36:37 AM
Log	Executing: tst_7_scrollModeGearbox	Aug 13, 2019 9:36:38 AM
Pass	ModeGearboxCentral/ModeGearboxSportCentral: Screenshot comparison of 'panelCentralConsoleIndication' passed	Aug 13, 2019 9:36:38 AM
Pass	ModeGearboxCluster/ModeGearboxSportCluster: Screenshot comparison of 'vwuGearboxValueWithUnit' passed	Aug 13, 2019 9:36:40 AM
Pass	ModeGearboxCentral/ModeGearboxRaceCentral: Screenshot comparison of 'panelCentralConsoleIndication' passed	Aug 13, 2019 9:36:41 AM
Pass	ModeGearboxCluster/ModeGearboxRaceCluster: Screenshot comparison of 'vwuGearboxValueWithUnit' passed	Aug 13, 2019 9:36:42 AM
Pass	ModeGearboxCentral/ModeGearboxSportCentral: Screenshot comparison of 'panelCentralConsoleIndication' passed	Aug 13, 2019 9:36:44 AM
Pass	ModeGearboxCluster/ModeGearboxSportCluster: Screenshot comparison of 'vwuGearboxValueWithUnit' passed	Aug 13, 2019 9:36:45 AM
Pass	ModeGearboxCentral/ModeGearboxComfortCentral: Screenshot comparison of 'panelCentralConsoleIndication' passed	Aug 13, 2019 9:36:46 AM
Pass	ModeGearboxCluster/ModeGearboxComfortCluster: Screenshot comparison of 'vwuGearboxValueWithUnit' passed	Aug 13, 2019 9:36:48 AM
Pass	ModeGearboxCentral/ModeGearboxSportCentral: Screenshot comparison of 'panelCentralConsoleIndication' passed	Aug 13, 2019 9:36:49 AM
Pass	ModeGearboxCluster/ModeGearboxSportCluster: Screenshot comparison of 'vwuGearboxValueWithUnit' passed	Aug 13, 2019 9:36:50 AM
Pass	ModeGearboxCentral/ModeGearboxRaceCentral: Screenshot comparison of 'panelCentralConsoleIndication' passed	Aug 13, 2019 9:36:51 AM
Pass	ModeGearboxCluster/ModeGearboxRaceCluster: Screenshot comparison of 'vwuGearboxValueWithUnit' passed	Aug 13, 2019 9:36:53 AM
Log	Memory usage: 378.21484375MB	Aug 13, 2019 9:36:54 AM
Log	Process ID: 20078	Aug 13, 2019 9:36:54 AM
Log	Memory usage: 169.765625MB	Aug 13, 2019 9:36:54 AM
Log	Process ID: 20155	Aug 13, 2019 9:36:54 AM
Log	Memory usage: 144.03515625MB	Aug 13, 2019 9:36:54 AM
Log	Process ID: 20209	Aug 13, 2019 9:36:54 AM
Log	Executing: tst_8_clickParkingBrake	Aug 13, 2019 9:36:55 AM
Pass	ParkingBrake/ParkingBrakeOn: Screenshot comparison of 'image4Image' passed	Aug 13, 2019 9:36:56 AM
Pass	ParkingBrake/ParkingBrakeOff: Screenshot comparison of 'image4Image' passed	Aug 13, 2019 9:36:57 AM
Pass	ParkingBrake/ParkingBrakeOn: Screenshot comparison of 'image4Image' passed	Aug 13, 2019 9:36:58 AM
Log	Memory usage: 378.265625MB	Aug 13, 2019 9:36:59 AM
Log	Process ID: 20078	Aug 13, 2019 9:36:59 AM
Log	Memory usage: 169.796875MB	Aug 13, 2019 9:36:59 AM
Log	Process ID: 20155	Aug 13, 2019 9:36:59 AM
Log	Memory usage: 144.04296875MB	Aug 13, 2019 9:36:59 AM
Log	Process ID: 20209	Aug 13, 2019 9:36:59 AM

Slika 5.40. *Rezultat izvođenja sustavskog testa*

6. ZAKLJUČAK

U ovom diplomskom radu koji je nastao u suradnji sa tvrtkom Rimac Automobili d.o.o., u odjelu ICU Software Development pod vodstvom Tomislava Lugarića, izrađena je aplikacija koja je omogućila testiranje sustava bez fizičkog prisustva mehaničkih ulaza na centralnoj konzoli informacijsko zabavnog sustava. Uz pomoć te aplikacije odrađeno je automatizirano testiranje grafičkog korisničkog sučelja te su pokazane brojne mogućnosti danih alata i tehnologija, ali uočeni i neki nedostatci. Provedeni su testovi jedinica, integracijskog testiranja i sustavskog testiranja koji su prošli u skladu s očekivanim vrijednostima.

Kao bitan čimbenik valja napomenuti kako je za pisanje kvalitetnog programskog koda aplikacije prema standardu potrebno barem znanje programskog jezika C++ koje se proširuje nadogradnjama Qt-a. Nadalje, kako bi testiranje bilo uspješno potrebno je poznavati i testno okruženje Qt alata. Međutim, to je i namjena ovoga alata jer se testiranje jedinica provodi od razvojnog programera dok su više razine testiranja zadatak osoba specijaliziranih za testiranje.

Alat Squish omogućuje pisanje testova u nekom od brojnih podržanih skriptnih programskih jezika, vrlo brza je linija učenja i korištenja alata što omogućuje vrlo brzo i jednostavno automatiziranje testiranja koja su popraćena brojnim primjerima unutar zajednice korisnika ovoga alata. Tehnologija asinkrone komunikacije ZMQ omogućuje ovakvu izvedbu također lakšom, odnosno omogućuje rad s aplikacijama izgrađenim za različite inačice. Neki od nedostataka korištenih alata su primjerice dosta dulja linija učenja za pisanje testova jedinica koristeći QtTest i QtQuickTest unutar alata Qt Creator, dok alat Squish uz svoje brojne podržane skriptne jezike pokazuje nedostatke u smislu zastarjelih i neažuriranih inačica programskih jezika.

7. LITERATURA

[1] Tiffiny Rossi, A brief history of in-vehicle infotainment and car data storage, https://www.tuxera.com/blog/a-brief-history-of-in-vehicle-infotainment-how-tuxera-fits-in/ (posjećeno 22.6.2019)

[2] tryQa, What is V-model- advantages, disadvantages and when to use it?, http://tryqa.com/what-is-v-model-advantages-disadvantages-and-when-to-use-it/ (posjećeno 23.6.2019.)

[3] Basics of Software Testing, A course by Quality House, version 3.1 July 2011

[4] Ronald Tyson, Tehcodebit, Software Engineering | SDLC V-Model, https://medium.com/techcodebit/software-engineering-sdlc-v-model-d4027e792ee3 (posjećeno 23.6.2019.)

[5] Wikipedia, V-Model (software development), https://en.wikipedia.org/wiki/V-Model_(software_development) (posjećeno 23.6.2019.)

[6] STC Admin, Difference Between Static Testing And Dynamic Testing, http://www.softwaretestingclass.com/difference-between-static-testing-and-dynamic-testing/ (posjećeno 24.6.2019.)

[7] Wikipedia, Test automation, https://en.wikipedia.org/wiki/Test_automation (posjećeno 24.6.2019.)

[8] Mercedes-Benz, Electric and Electronic Components in Motor Vehicles up to 3,5t – General Requirements, Test Conditions and Tests Part I: Electrical Requirements and Tests 12 V On-Board Electrical System

[9] Mercedes-Benz, Electric and Electronic Components in Motor Vehicles up to 3,5t – General Requirements, Test Conditions and Tests. Part 2: Environmental Requirements

[10] Barbara J. Czerny, Joseph D'Ambrosio, Rami Debouk, General Motors Research and Development Kelly Stashko, General Motors Powertrain, ISO 26262 Functional Safety Draft International Standard for Road Vehicles: Background, Status, and Overview

[11] Com-power Corporation, What's the Difference Between EMI and EMC?, https://www.com-power.com/emi-emc-differences.html (posjećeno 27.6.2019.)

[12] MPL High-Tech, IP Ratings (Ingress Protection), https://www.mpl.ch/info/IPratings.html (posjećeno 27.6.2019.)

[13] Ranorex GmBH, Test Automation for All, https://www.ranorex.com/ (posjećeno 25.6.2019)

[14] Ranorex GmBH, Studio start page, https://www.ranorex.com/rx-media/graphics/startpage_slider-testsuite.png (posjećeno 25.06.2019.)

[15] Ranorex GmBH, Studio features, https://www.ranorex.com/features/ (posjećeno 25.06.2019.)

[16] Ranorex GmBH, Qt testing, https://www.ranorex.com/help/latest/interfaces-connectivity/technology-instrumentation/qt-testing/ (posjećeno 25.6.2019)

[17] Wikipedia, Squish (Froglogic), https://en.wikipedia.org/wiki/Squish_(Froglogic) (posjećeno 25.06.2019.)

[18] Froglogic GmBH youtube kanal, Embedded Device GUI Testing, https://www.youtube.com/watch?v=2jr5rJic414 (posjećeno 25.06.2019.)

[19] Froglogic, Squish features, https://www.froglogic.com/squish/features/ (posjećeno 25.06.2019)

[20] Wikipedia, Squish (Froglogic), https://en.wikipedia.org/wiki/Squish_(Froglogic) (posjećeno 25.06.2019)

[21] Froglogic, Automated Testing of Embedded Qt HMIs, https://www.froglogic.com/squish/editions/automated-testing-embedded-qt-guis-hmis/ (posjećeno 25.06.2019)

[22] Lenovo, ThinkPad T470p Platform Specifications, https://psref.lenovo.com/syspool/Sys/PDF/ThinkPad/ThinkPad%20T470p/ThinkPad_T470p_Platform_Specifications.pdf (posjećeno 26.06.2019.)

[23] Ubuntu, Ubuntu 18.04.2 LTS (Bionic Beaver), http://releases.ubuntu.com/18.04/ (posjećeno 26.06.2019.)

[24] Ubuntu wiki, Bionic Beaver ReleaseNotes, https://wiki.ubuntu.com/BionicBeaver/ReleaseNotes (posjećeno 26.06.2019.)

[25] Qt5 Cadaques, J. Ryannel, J.Thelin, Release 2015-03

[26] Qt wiki, About Qt, https://wiki.qt.io/About_Qt (posjećeno 27.06.2019)

[27] Qt Company, All Modules, https://doc.qt.io/qt-5/qtmodules.html (posjećeno 27.06.2019)

[28] Qt Company, Qt for Device Creation, https://doc.qt.io/QtForDeviceCreation/ (posjećeno 27.06.2019)

[29] Qt Company, Boot to Qt Software Stack, https://doc.qt.io/QtForDeviceCreation/qtb2-index.html (posjećeno 27.06.2019)

[30] Qt wiki, Language Bindings, https://wiki.qt.io/Language_Bindings (posjećeno 27.06.2019)

[31] Qt Company, Qt Creator quick tour, https://doc.qt.io/qtcreator/creator-quick-tour.html (posjećeno 27.06.2019)

[32] Guillaume Lazar, Robin Penea, Mastering Qt5, Master application development by writing succinct, robust and reusable code with Qt 5, 2016.

[33] Qt Company, Qt Test Overview, https://doc.qt.io/qt-5/qtest-overview.html (posjećeno 27.06.2019)

[34] Qt Company, Qt Test https://doc.qt.io/qt-5/qttest-index.html (posjećeno 27.06.2019)

[35] Qt Company, Qt Quick Test, https://doc.qt.io/qt-5/qtquicktest-index.html (posjećeno 16.08.2019)

[36] Squish testing, https://www.froglogic.com/squish/

[37] Image comparison modes in screenshot verification points, https://kb.froglogic.com/display/KB/Image+comparison+modes+in+screenshot+verification+points (posjećeno 01.09.2019.)

[38] Correlation comparison mode, https://doc.froglogic.com/squish/latest/ide.dialogs.html#ide.setting.comparison.mode.correlation (posjećeno 01.09.2019.)

[39] Pieter Hintjens, ØMQ - The Guide, http://zguide.zeromq.org/page:all (posjećeno 27.06.2019.)

SAŽETAK

U ovom radu opisana je primjena automatiziranog testiranja nad informacijsko-zabavnim sustavom vozila Rimac Concept_One. Prvo je izrađena aplikacija koja koristi ZMQ način komunikacije *publish-subscribe*, a služi za simuliranje mehaničkih ulaza korisnika prema zaslonima vozila. Zatim je nad takvim sustavom kroz integrirano razvojno okruženje Qt Creator izvedeno testiranje jedinica metodom podatcima vođenog testiranja na C++ izvornom kodu te QML grafičkim sučeljima aplikacija vozila. Svi rezultati provedenih testiranja u skladu su sa očekivanjima. Alat Squish korišten je za automatizaciju provedbe integracijskih testova te sustavskog testiranja primjenom metode snimaka zaslona s korelacijskim modelom podudaranja, a dobiveni rezultati također su u skladu sa očekivanima.

Ključne riječi: automatizirano testiranje, informacijsko-zabavni sustav, Qt Creator, Squish, ZMQ

ABSTRACT

Automated in-vehicle infotainment testing

This paper describes the application of automated testing over in-vehicle infotainment system for vehicle Rimac Concept_One. Firstly, an application that uses ZMQ publish-subscribe communication method has been created to simulate mechanical user inputs to vehicle screens. Then, through the Qt Creator integrated development environment, unit testing was performed on the C++ source code and QML graphical user interface. All of the test results were as expected. The Squish tool was used to automate integration and system testing using a screenshot method with a correlation model of matching and the obtained results are also in line with the expected ones.

Keywords: automated testing, in-vehicle infotainment, Qt Creator, Squish, , ZMQ

ŽIVOTOPIS

Leo Matančić rođen je 18. siječnja 1996. godine u Virovitici. Završio je Osnovnu školu Ivane Brlić Mažuranić u Virovitici, te upisao prirodoslovno-matematički smjer u Gimnaziji Petra Preradovića u Virovitici. U srednjoj školi natjecao se u znanju na natjecanjima Rokovi matematičari te Klokan bez granica. Nakon završene gimnazije 2014. godine upisuje preddiplomski studij računarstva na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijek. Po završetku preddiplomskog studija 2017.godine upisuje diplomski studij Automobilsko računarstvo i komunikacija na Fakultetu elektrotehnike, računarstva i informacijskih tehnologija u Osijeku.