OULUN AMMATTIKORKEAKOULU

Sakari Klasila

FYYSINEN TESTAUSROBOTTI MOBIILITESTIAUTOMAATION TUKENA

FYYSINEN TESTAUSROBOTTI MOBIILITESTIAUTOMAATION TUKENA

Sakari Klasila
Opinnäytetyö
Kevät 2023
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, ohjelmistokehitys

Tekijä: Sakari Klasila
Opinnäytetyön nimi: Fyysinen testausrobotti mobiilitestiautomaation tukena
Työn ohjaaja: Olli Himanka
Työn valmistumislukukausi ja -vuosi: Kevät 2023 Sivumäärä: 30 + 9 liitettä

Tämän opinnäytetyön tarkoituksena oli suunnitella ja toteuttaa fyysinen testausrobotti mobiilitestiautomaation tueksi. Robottia ei kehitetty korvaamaan muita automaatiotyökaluja, vaan sen tarkoituksena on avustaa mobiilitestiautomaatiota tilanteissa, joissa tarvitaan testilaitteen kosketusnäytön manipulointia fyysisin kosketuksin.

Opinnäytetyön aihe tuli tietyltä yritykseltä, mutta tämä ratkaisumalli voi hyödyttää myös muita samankaltaisissa tilanteissa olevia tahoja. Opinnäytetyö päätettiin toteuttaa yleisluontoisesti, joten siitä jätettiin pois joitain tunnistettavia sovellus- ja testiympäristökohtaisia yksityiskohtia. Työn alussa kerrottiin yleisesti mobiilitestiautomaatiosta ja niistä työkaluista, joita käytettiin testausrobotin ja valmiin testiympäristön kehittämisessä.

Robotin toteutuksessa hyödynnettiin 3D-tulostinta, jota muokattiin niin, että sen ohjaaminen oli mahdollista testiympäristöstä käyttäen vakiintuneita laiteohjausmetodeja ja G-koodi-koneohjauskieltä. Testiympäristöön luotiin uusi Python-kirjasto, johon määriteltiin ohjausmetodit, joita hyödyntäen robottia voitiin käyttää saumattomasti osana testausympäristöä.

Valmista testausrobottia testattiin luomalla sille mobiilitestausympäristö, joka sisälsi kaikki tarvittavat ominaisuudet sen toimintojen varmistamiseksi. Testilaitteina käytettiin neljää mobiililaitetta, joille testattava sovellus asennettiin. Testiympäristöön luotiin 20 identtistä testitapausta, jotka suoritettiin satunnaisesti näillä testilaitteilla. Testausrobotti suoritti kaikkien testitapausten kirjautumisvaiheet onnistuneesti.

Lopputuloksena saatiin toimiva fyysinen testausrobotti, joka toimii määrätyllä tavalla osana mobiilitestausympäristöä. Varmennustestaus osoitti laitteen luotettavuuden, ja testausrobotti otettiin heti käyttöön osaksi testausympäristöä yritykselle, joka oli aiheen alkuperäinen tarpeenasettaja.

Asiasanat: testiautomaatio, robotiikka, laiteohjaus, testaaminen, automaatio, mobiilisovellukset

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Tehcnology, Option of Software Development

---

Author: Sakari Klasila
Title of thesis: Physical testing robot to support mobile test automation
Supervisor: Olli Himanka
Term and year when the thesis was submitted: Spring, 2023
Number of pages: 30 + 9 appendices

This thesis aimed to design and implement a physical testing robot to support mobile test automation. The robot was not meant to replace other automation tools but to assist in situations where physical touch manipulation of the test device's touchscreen is required

The thesis provided an overview of mobile test automation and the tools used in the development of the testing robot and the test environment. The robot was implemented using a 3D printer, which was modified to be controlled from the test environment using established device control methods and G-code machine control language. A new Python library was created for the robot within the test environment, defining control methods that enabled its seamless integration into the testing environment.

The completed testing robot was tested by creating a mobile testing environment specifically designed to ensure all its functionalities. Four mobile devices were used as test devices, with the tested application installed on them. Twenty identical test cases were created in the test environment and randomly executed on each test device. The testing robot successfully completed all the login steps for each test case.

As a result, a functional physical testing robot was achieved, operating as intended as part of the mobile testing environment. Verification testing demonstrated the device's reliability, and the testing robot was immediately put into use as part of the testing environment for the company that had the initial need for it.

Keywords: test automation, robot, device control, testing, automation, mobile applications

SISÄLLYS

1 JOHDANTO

Mobiilisovellusten kehittäjät ja julkaisijat käyttävät usein kolmansien osapuolien tunnistussovelluksia varmistaakseen asiakkaiden pääsyn palveluihinsa. Samoin tekee eräs yritys, jonka tarpeesta tämän opinnäytetyön aihe syntyi. Yrityksen testiautomaatio on tähän asti toteutettu verkko-ohjainpohjaisilla työkaluilla, jotka mahdollistavat testattavan sovelluksen ohjaamisen komentosarjojen avulla ilman tarvetta fyysiselle vuorovaikutukselle testilaitteen kosketusnäytön kanssa.

Yrityksen käyttämän kolmannen osapuolen tunnistussovelluksen kehittäjä on estänyt sovelluksen ohjelmallisen manipuloinnin. Sovelluksen kirjautumisvaiheessa vaadittavan PIN-koodin syöttäminen edellyttää fyysistä kosketusta laitteen kosketusnäyttöön, mikä tarkoittaa sitä, että yrityksen käyttämien komentosarjapohjaisten automaatiotyökalujen hyödyntäminen sen automatisoinnissa ei ole mahdollista. Testattava mobiilisovellus vaatii uudelleenkäynnistyksen ja sisäänkirjautumisen jokaisen testitapauksen välissä, joten kirjautumisprosessin automatisointi on kriittisen tärkeää testien suorittamisen kannalta.

Tämän opinnäytetyön tarkoituksena on kuvata fyysisen testausrobotin kehittämistä mobiilitestiautomaation tueksi. Testausrobotin tavoitteena ei ole korvata muita automaatiotyökaluja, vaan päivittää testiautomaatioympäristö tukemaan robotin käyttöä silloin, kun se on tarpeellista. Testitapausten muut vaiheet suoritetaan perinteistä komentosarjapohjaista automaatiota hyödyntäen.

Valmis testausrobotti toimii simuloituna ihmiskäyttäjänä, joka syöttää mobiilisovelluksen kirjautumisvaiheessa PIN-koodin laitteeseen kosketusnäytön kautta. Tämä mahdollistaa testiautomaatioprosessin sujuvan etenemisen ilman keskeytyksiä. Koko testauksen toteuttaminen fyysisen robotin avulla ei ole tarpeellista, sillä perinteiset komentosarjapohjaiset automaatiotyökalut soveltuvat lähes poikkeuksetta mobiilisovellusten testaamiseen

Tässä opinnäytetyössä ei ole tarpeellista eritellä tiettyjä sovelluksia tai niiden julkaisijoita, sillä ratkaisu on yleisluontoinen ja sitä voi hyödyntää vastaavissa tapauksissa myös muiden testiautomaatioratkaisujen yhteydessä.

2 MOBIILISOVELLUSTEN AUTOMAATTINEN TESTAAMINEN

Testiautomaatio on vakiintunut käytäntö ohjelmistoalan yritysten testausprosesseissa, olipa kyseessä mobiilisovellusten tai muiden ohjelmistoratkaisujen kehittäminen. Mobiilisovellusten automaattinen testaus vähentää manuaalisen työn tarvetta, mikä helpottaa erityisesti suurten testimassojen suorittamista usein ja säännöllisesti. Hyvin suunniteltu automaatiotestaus suorittaa testit nopeammin kuin ihmiskäyttäjä ja aina samalla tarkkuudella. (1, sivut 3-4.)

Mobiilitestiautomaation toteuttaminen nykyaikaisilla työkaluilla tarjoaa monipuolisia ratkaisuja lähes kaikkiin testaustarpeisiin. On olemassa useita eri alustariippumattomia testaustyökaluja, joita voidaan käyttää testiautomaation suorittamiseen useilla eri laitteilla yhtäaikaisesti ja riippumatta käyttöjärjestelmästä. Esimerkiksi Robot Framework on alustariippumaton testiautomaatiokehys, joka tarjoaa työkalut kaikkien laitteiden ja sovellusten testaamiseen. (2.)

Vaikka testiautomaatioon panostaminen yleensä kannattaa, kaikkea ei välttämättä kannata automatisoida. Voi olla esimerkiksi kannattavampaa suorittaa manuaalisesti sellaiset testitapaukset, jotka suoritetaan vain kerran tai hyvin harvoin. Sen lisäksi käyttökokemustestauksen automatisointi voi olla haasteellista, koska sovelluksen käyttökokemus on suunniteltu ihmiskäyttäjille eikä sen automatisointi ole välttämättä mahdollista. Testiautomaation suunnittelu vaatii alkuinvestointeja ja huolellista suunnittelua, mutta pitkällä aikavälillä se voi tuottaa merkittäviä etuja yrityksille. (1, sivu 3.)

2.1 Mobiilisovellusten testiympäristö

Testiympäristö on kokonaisuus, joka käsittää kaikki tarvittavat resurssit, työkalut ja menetelmät testien suorittamiseen. Se voi sisältää esimerkiksi tietokoneen, testaustyökaluja, testilaitteita sekä sopivat työtilat testausta varten. Testiympäristö voi koostua useista kohdennetuista ympäristöistä eri testausvaiheisiin tai -tyyppeihin, kuten yksikkötestiympäristö tai suoritustestiympäristö. Se voi myös sisältää useita keskenään liitettyjä järjestelmiä tai virtuaaliympäristöjä. (3, luku 3.184.)

Mobiilitestiautomaation yhteydessä testiympäristö voi sisältää esimerkiksi tietokoneen, johon on asennettu tarvittavat testausohjelmat, kuten testikehys ja viestintäprotokollat tietokoneen ja testilaitteiden välillä. Testilaitteina voi käyttää joko fyysisiä mobiililaitteita tai emulaattoreita, joihin testattava sovellus on asennettu.

Mobiilitestaus ei yleensä edellytä fyysisiä automaatioratkaisuja, kuten fyysistä robottia, testilaitteiden tai -sovellusten ohjaamiseen. Sen sijaan usein käytetään automaatiotyökaluja, jotka hyödyntävät laitteiden välisiä verkko-ohjainprotokollia. Esimerkiksi Appium on automaatiotyökalu, joka lähettää komentosarjoja mobiililaitteen verkko-ohjaimelle ja saa laitteen suorittamaan erilaisia toimintoja, kuten painikkeiden painamista ja näytön lukemista. (4.)

2.2 Mobiilitestiautomaation työkalut

2.2.1 Robot Framework

Testiautomaatiokehys ohjaa testiprosessien kulkua ja tarjoaa rakenteen ja työkalut testien suorittamiseen. Yksi suosittu testiautomaatiokehys on Python-pohjainen ja avoimen lähdekoodin Robot Framework. Se on suunniteltu tukemaan alustariippumatonta testiautomaatiota ja ohjelmistorobotiikan kehittämistä. Robot Framework on laajalti käytetty testiautomaatiotyökalu sekä pienissä että suurissa yrityksissä ympäri maailman. (2.)

Paul Laihosen artikkelin "Robot Framework: Menneisyys, Nykyhetki ja Tulevaisuus" mukaan Robot Frameworkin kehitys alkoi vuonna 2005 suomalaisen Pekka Klärkin diplomityöstä. Sen kehittämistä jatkoi samana vuonna Nokia Networks, joka julkaisi siitä avoimen lähdekoodin version vuonna 2008. (5.)

Nykyään Robot Frameworkia ylläpitää voittoa tavoittelematon konsortio nimeltä Robot Framework Foundation, jonka tavoitteena on varmistaa automaatiokehyksen kehittämisen jatkuminen ja sen saatavuus tulevaisuudessa. Robot Frameworkin virallisten verkkosivujen mukaan Robot Framework Foundationin jäseniin kuului vuonna 2022 ainakin 49 yritystä, mukaan lukien Nokia, Nordea, Finnair ja Veikkaus. (6.)

Robot Frameworkin syntaksi koostuu yksinkertaisista avainsanoista (keywords), joiden avulla testitapaukset (test cases) luodaan. Tämän syntaksin ihmisluettavuus erottaa sen monista ohjelmointikielistä, ja sen helppokäyttöisyys mahdollistaa kehyksen käytön ja testitapausten luomisen myös ilman varsinaista ohjelmointikokemusta tai alan koulutusta. (2.)

Robot Frameworkissa testitapaukset kirjoitetaan robot-tiedoston Test Cases -osion alle (kuva 1). Testitapaukset koostuvat avainsanoista (kuva 2), jotka voivat olla määritetty joko itse luoduissa avainsanakirjastoissa tai tuotu muista kirjastoista, kuten Builtin ja AppiumLibrary, jotka on kehitetty Robot Frameworkia varten. (3.)

```
14  *** Test Cases ***
15
16  Login to Mobile App And Verify Home Screen
17      [Tags]  Login
18      Wait Until Login Page Is Displayed
19      Open Authentication Application
20      Login With Pincode Using Test Robot
21      Verify That Home Screen Is Displayed
```

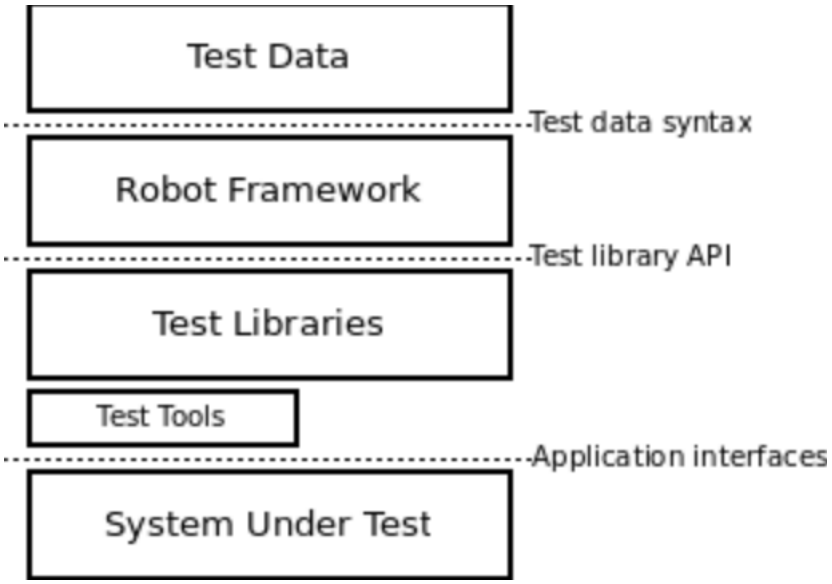
Kuva 1. Esimerkki Robot Frameworkilla luodusta helposti testitapauksesta.

```
18      Click Element       ${TEXT_INPUT_FIELD}
19      Input Text          text    ${TEXT_INPUT_FIELD}
```

Kuva 2. Esimerkki avainsanojen Click Element ja Input Text käyttämisestä Robot Frameworkin testitapauksessa.

Robot Frameworkin verkkosivuille on listattuna suosituimmat sisäiset ja ulkoiset kirjastot, jotka helpottavat testikehyksen käyttöä (2). Esimerkiksi suosituin Selenium2Library-kirjasto sisältää valmiiksi määriteltyjä avainsanoja etenkin verkkosivutestaamiseen (7) ja mobiilitestaukseen on tarjolla kattava AppiumLibrary-kirjasto, joka sisältää avainsanoja mobiilisovellusten testaamiseen ja korkean tason metodit mobiililaitteiden kanssa kommunikoimiseen (8). Robot Framework mahdollistaa myös muiden ohjelmointikielten, kuten Pythonin ja Javan, kehittämien kirjastojen saumattoman käytön osana testiympäristöä (2).

Robot Frameworkin modulaarinen arkkitehtuuri (kuva 3) mahdollistaa testikehyksen skaalautuvuuden ja sopivuuden kaikkiin testaustarpeisiin (9.). Jotkut kirjastot käyttävät ulkoisia työkaluja testaamisen tukena, kuten mobiililaitteiden ja mobiilisovellusten testaamiseen kehitetty AppiumLibrary-kirjasto. AppiumLibrary käyttää Appium -testiautomaatiotyökalua testikehyksen ja testilaitteen välisen viestinnän mahdollistavana menetelmänä. (10, sivu 16.)

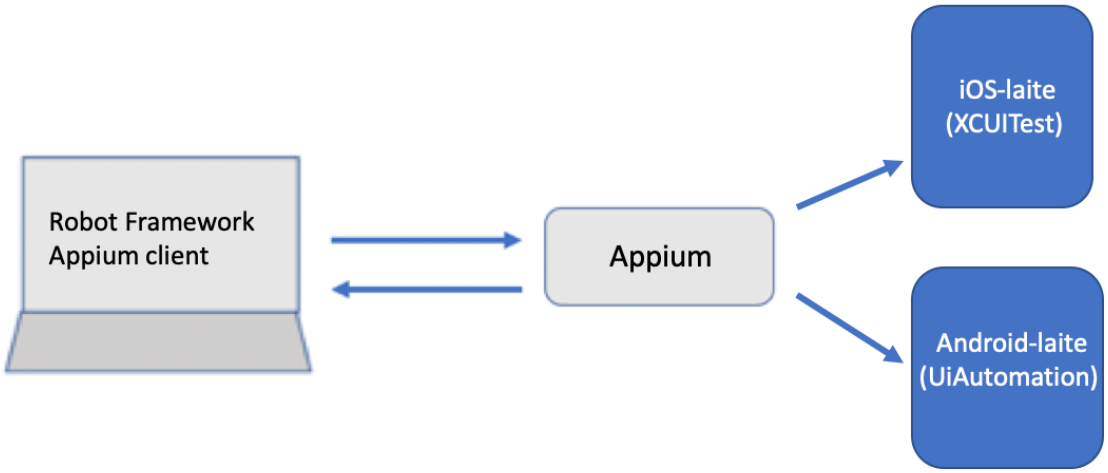


Kuva 3. Robot Frameworkin arkkitehtuuri. (11.)

2.2.2 Appium

Appium on avoimen lähdekoodin automaatiotyökalu, joka on suunniteltu natiivi-, hybridi- ja mobiiliverkkosovellusten sekä työpöytäsovellusten testaamiseen. Se on toteutettu Node.js-ympäristössä ja toimii verkkopalvelimen tavoin tarjoten rajapinnan mobiililaitteiden hallintaan testikehyksen kautta. Appium lähettää komentosarjoja mobiililaitteille, mikä mahdollistaa mobiililaitteiden oman

testikehyksen käytön sovellusten ohjaamiseen. Appium toimii viestintämenetelmänä testikehyksen, kuten Robot Frameworkin, ja testilaitteiden välillä (kuva 4) (10, sivu 16.).



Kuva 4. Appium toimii viestintätyökaluna testikehyksen ja mobiililaitteiden välillä.

Appium käyttää mobiililaitteiden omia verkko-ohjaimia, kuten UiAutomatoria Android-laitteille ja XCUITestia iOS-laitteille, kommunikoidakseen mobiilisovellusten kanssa. Tämä mahdollistaa alustariippumattoman testiautomaatioympäristön kehittämisen sekä iOS- että Android-käyttöjärjestelmille kehitettyjen sovellusten testaamiseen (10, sivut 16-17.).

2.2.3 AppiumLibrary

AppiumLibrary on mobiilisovellusten testaamista varten kehitetty kirjasto, joka integroituu saumattomasti Robot Framework -testiautomaatiokehykseen. Tämä kirjasto tarjoaa Robot Frameworkille korkean tason metodit kommunikointiin testilaitteiden kanssa sekä valmiita avainsanoja, jotka helpottavat testitapausten luomista (8.).

AppiumLibrary-kirjaston avainsanoja käytetään Robot Frameworkissa samalla tavalla kuin muitakin avainsanoja. Esimerkiksi avainsana "Open Application" käynnistää testattavan sovelluksen mobiililaitteessa, kun taas avainsana "Close Application" sulkee sovelluksen testin päätyttyä (8.). Kirjaston avainsanoja voidaan käyttää testitapauksissa sovelluksen hallintaan ja sen toimintojen testaamiseen.

3 TESTAUSROBOTIN TOTEUTUS

Tässä luvussa esitellään fyysisen testausrobotin toteutuksen vaiheita mobiilitestiautomaation tueksi. Testausrobotin lisäksi testiympäristöön kuului neljä Android-laitetta, joihin testattava sovellus ja tunnistussovellus asennettiin. Testiympäristöön kuului myös Mac-tietokone, johon oltiin asennettu tarvittavat automaatiotyökalut. Testiautomaatiokehyksenä käytettiin Robot Frameworkia.

Mobiilisovelluksen testaamisessa hyödynnettiin Appium-verkko-ohjainprotokollaa (8.) ja Robot Frameworkin AppiumLibrary-kirjastoa. Fyysistä testausrobotti suunniteltiin toimimaan vain testitapauksen kirjautumisvaiheessa, jolloin sovelluksen ohjaaminen Appiumin lähettämien komentosarjojen avulla ei ole mahdollista. Robotin toimintaperiaate suunniteltiin niin, että kirjautumisvaiheessa se aktivoituu ja ohjaa testilaitetta fyysisesti koskettamalla sen näyttöä ja kirjaa käyttäjän sisään. Tämän jälkeen testiympäristö jatkaa testitapauksen suorittamista Appium-automaatiotyökalua hyödyntäen.

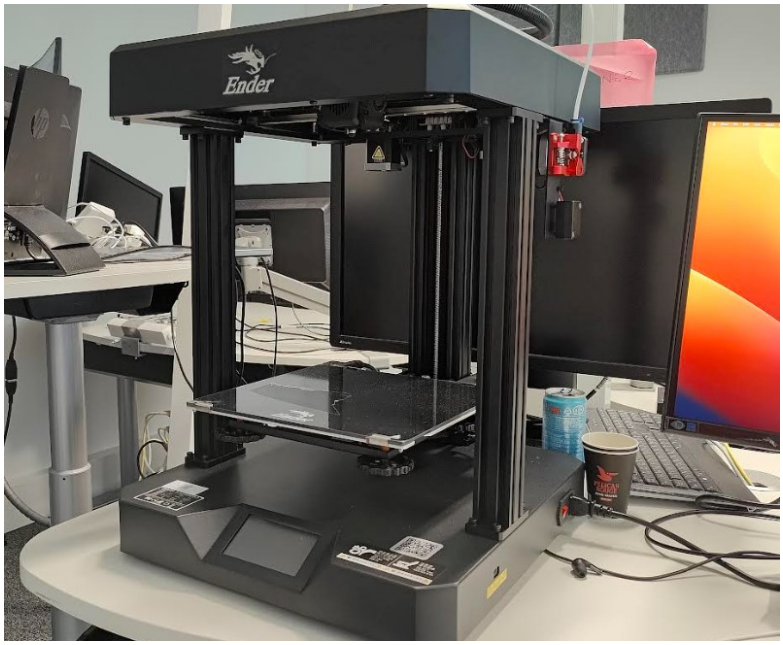
3.1 Toteutustavan valinta

Testausrobotille määriteltiin muutama vaatimus: Robotin on oltava helposti muokattavissa, ohjattavissa jollain yleisellä koneohjauskielellä ja siihen kehitettävälle testausalustalle pitää mahtua neljä mobiililaitetta testauskäyttöön. Laitteen ei pitäisi myöskään vaatia ylimääräisten ohjelmistojen lataamista testejä suorittavalle tietokoneelle. Ylimääräisiksi ohjelmistoiksi ei lasketa esimerkiksi yleisiä python-kirjastoja, mutta erillisten ajureiden tai mahdollisen muokattavan laitteen mukana tulevien ohjelmistojen lataamisen ei pitäisi olla pakollista.

Heti alussa kiinnostuin 3D-tulostimista ja niiden käyttämisestä robotin perustana. Huomasin, että 3D-tulostimet ovat yleensä kevyitä ja helposti muokattavissa, koska tulostimen fyysiset osat voidaan yleensä vaihtaa ja irrottaa helposti. Yleisimpiä 3D-tulostimia ohjataan G-koodi-koneohjauskielellä, joka on vakiintunut CNC-laitteiden ohjauksessa. G-koodikomennoilla ohjattaville laitteelle voidaan syöttää ohjauskomentoja esimerkiksi tietokoneelta USB-johdon kautta. (11, luku 5.)

Markkinoilla on runsaasti erilaisia 3D-tulostimia, ja useimmat niistä toimivat samankaltaisella periaatteella. Suurimmat erot tulostimien välillä liittyvät tulostuslaatuun, liikkeiden tarkkuuteen ja tulostusnopeuteen. Tekniset erot eivät kuitenkaan vaikuta merkittävästi laitteen soveltuvuuteen tähän käyttötarkoitukseen, sillä lähes kaikki saatavilla olevat tulostimet täyttäisivät testausrobotille asetetut vaatimukset. Valitsin tulostimen yrityksen sisäisen kanavan kautta, koska yksi suosituimmista 3D-tulostinmalleista oli saatavilla yhteistyökumppanimme valikoimassa, ja olin vakuuttunut siitä, että se sopii tarpeisiimme.

Päätin valita Creality Ender 7 -3D-tulostimen muokattavaksi laitteeksi erityisesti sen avoimen rakenteen ansiosta, mikä mahdollistaa testauksessa käytettävien mobiililaitteiden helpon käsittelyn sekä tulostimen tulostuspään muokkaamisen tarpeidemme mukaisesti. Ender 7 -tulostimen alusta on sopivan kokoinen, jotta siihen mahtuu useita mobiililaitteita samanaikaisesti. Laitetta voidaan ohjata tietokoneen SerialPortin ja USB-johdon avulla käyttäen G-koodikomentoja, eikä sen ohjaaminen vaadi erillisten ohjelmistojen asentamista tietokoneelle. Projektissa käytössäni oleva Mac-käyttöjärjestelmä sisältää tarvittavat ajurit tämän 3D-tulostimen ohjaamiseksi.

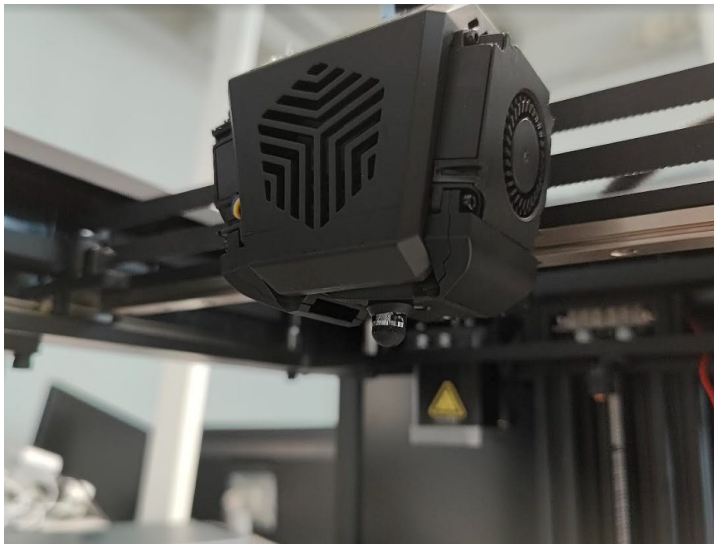


Kuva 5. Creality Ender 7 ennen muutoksia.

3.2 3D-tulostimen muuttaminen testausrobotiksi

Crealityn Ender 7 -3D-tulostimeen tehtiin tarvittavat fyysiset muutokset, jotka tekivät siitä testausrobotin. Tulostuspää korvattiin kosketuskynällä (kuva 6), joka toimii ihmiskäyttäjää mukailevana kosketuspäänä. Kosketuspäällä voidaan ohjata testauksessa käytettävää mobiililaitetta.

Lisäksi tulostimesta poistettiin materiaalinsyöttökomponentteja jäähdyttävä kovaääninen tuuletin. Tuulettimen poistaminen on perusteltua, koska se on kovaääninen ja oletusarvoisesti jatkuvasti päällä, mikä saattaisi aiheuttaa häiriötä erityisesti toimistoympäristössä. Osien poistaminen ei vaikuta robotin turvallisuuteen, koska tulostus- ja materiaalinsyöttötoimintoja ei käytetä mobiilisovellusten testauksessa.



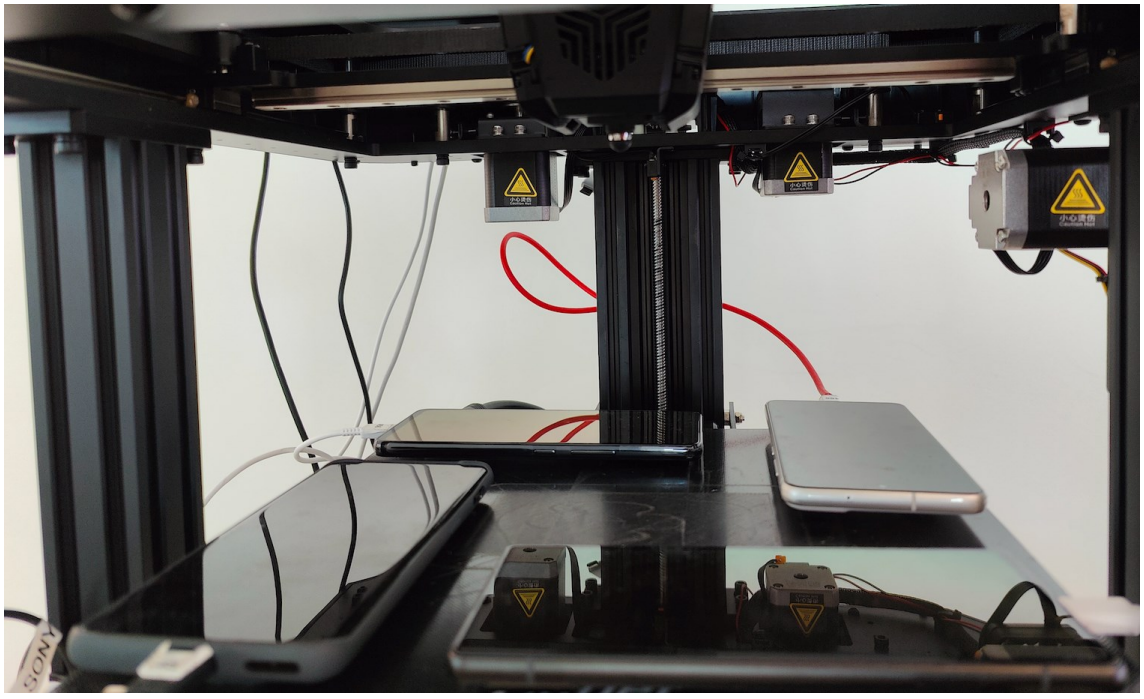
Kuva 6. Tulostimen tulostuspää korvattiin tavallisen kosketusnäyttökynän kärjellä.

Robottia ohjataan sarjaporttina toimivan USB-johdon kautta G-koodikomennoilla. Ohjaamista testattiin ennen ohjauslogiikan lisäämistä testiympäristöön lähettämällä yksinkertaisia G-koodimuotoisia testikomentoja laitteelle, jotta sen toiminnallisuudet saadaan varmistettua. G-koodikomennot lähetettiin laitteeseen käyttämällä Pythonia ja PySerial-kirjastoa (kuva 7). Kosketuspäätä ohjataan millimetrin tarkkuudella x-, y- ja z-akseleiden suuntaisesti. Luvussa 3.3. kerrotaan tarkemmin tulostimen liikuttamisesta G-koodikomentojen avulla ja sen ohjauslogiikasta.

```python
import serial
import time

# Establish serial connection with printer
ser = serial.Serial('/dev/tty.usbserial-21310', 115200)

# Home all axes
ser.write(b'G28\n')
time.sleep(5) # Wait for homing to finish

# Move to (50,40,20) at a feed rate of 3000 mm/min
ser.write(b'G1 X50 Y40 Z20 F3000\n')
time.sleep(10) # Wait for movement to finish

# Close serial connection
ser.close()
```

Kuva 7. G-koodikomennot lähetettiin 3D-tulostimelle Pythonin ja PySerial-kirjaston avulla.

Kun laitteiden välisen kommunikaation toimivuus varmistettiin, testausrobottiin suunniteltiin testausalusta mobiililaitteita varten. Alustassa on neljä paikkaa puhelimille, joiden sijainti alustalla määriteltiin laitekohtaisesti testiympäristössä. Puhelimet sijoitettiin laitteen reunoille siten, että testausrobotin kosketuspään ympärillä oleva kotelo mahtuu lepäämään alustan keskellä tarvittaessa (kuva 8).



Kuva 8. Testilaitteet asetettiin testausalustan reunoille niin, että kosketuspää koteloineen mahtuu tarvittaessa turvallisesti lepäämään alustan keskellä.

3.3 Robotin ohjaamiseen käytettävät työkalut

3.3.1 G-koodi-koneohjauskieli

G-koodi on koneohjauskieli, jota käytetään numeerisesti ohjattujen laitteiden, kuten useimpien 3D-tulostinten, jyrsinkoneiden ja sorvien, liikkeiden ja toimintojen ohjaamiseen. G-kooditiedosto koostuu sarjasta komentoja, joita koneen ohjain tulkitsee ja suorittaa. Näitä komentoja ovat esimerkiksi työkalun siirtäminen tiettyjä reittejä pitkin, nopeuden ja liikesuunnan säätäminen sekä muiden koneen toimintojen, kuten työkalun vaihdon, jäähdytyksen ja lämmityksen, ohjaaminen. (11, luku 6.)

3.3.2 PySerial-kirjasto ja sarjaporttiyhteys

PySerial on Python-kirjasto, joka mahdollistaa kommunikoinnin sarjaporttiyhteyden kautta erilaisten laitteiden, kuten 3D-tulostimen ja tietokoneen, välillä. Se hoitaa sarjaporttiyhteyden tekniset yksityiskohdat ja tarjoaa käyttöliittymän datan lähettämiseen ja vastaanottamiseen laitteiden välillä. (12.) Tässä projektissa PySerial-kirjastoa käytetään testausrobotin ohjaamiseen sarjaporttina toimivan USB-kaapelin kautta.

3.3.3 PyRobotControl-kirjaston luominen testausrobotin ohjaamiseen

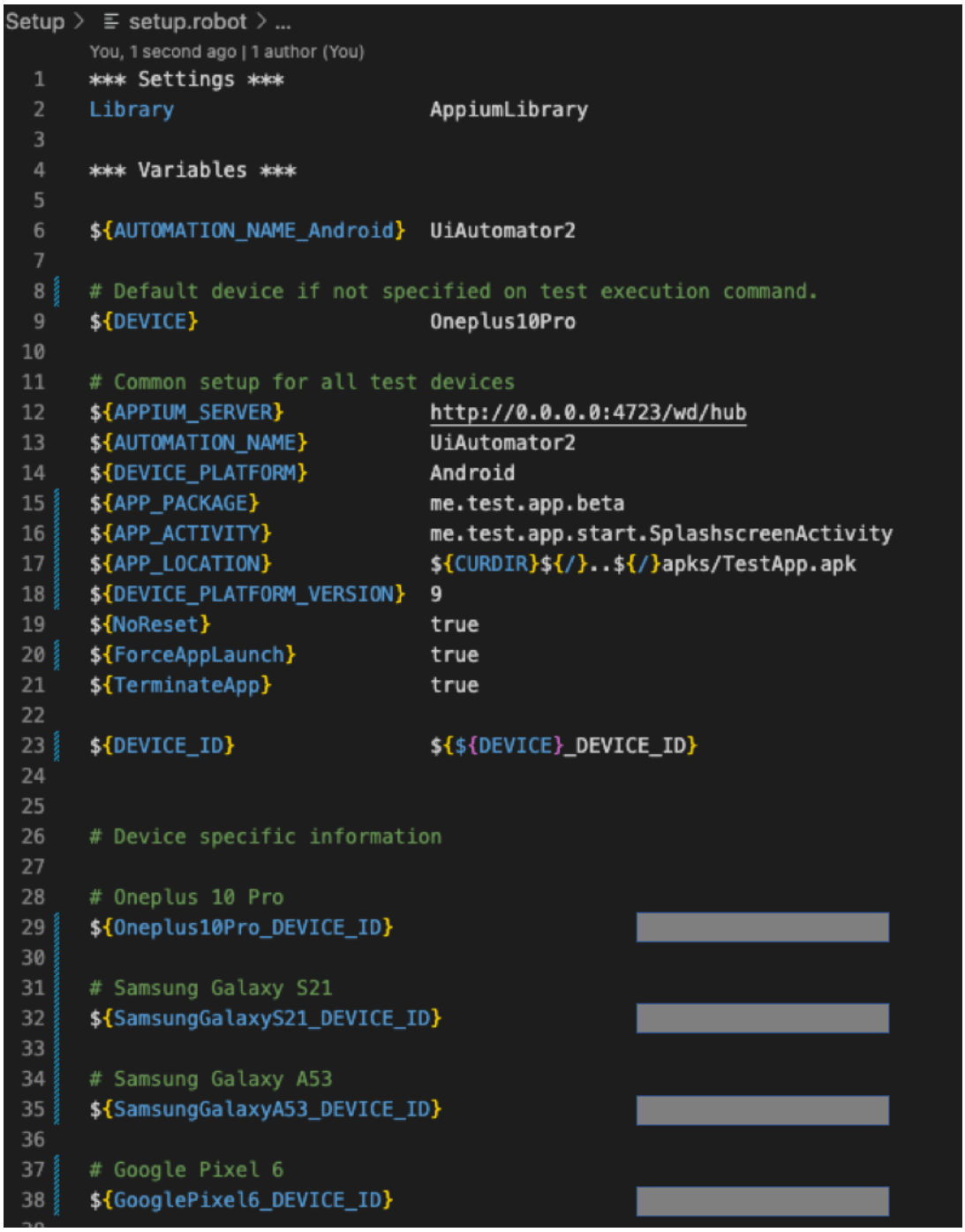
Testiympäristön testikehyksen ja testausrobotin saumattoman yhteistyön varmistamiseksi ympäristöön luotiin uusi Python-kirjasto nimeltä PyRobotControl (kuva 17.). Tämä kirjasto sisältää kaikki tarvittavat metodit G-koodikomentojen lähettämiseksi testausrobotille sarjaportin kautta Robot Frameworkista käsin. PyRobotControl.py-tiedoston koko sisältö on saatavilla liitteissä 1, 2 ja 3.

PyPrinterControl-kirjasto hyödyntää PySerial-kirjastoa, jonka metodit mahdollistavat kommunikoinnin testausrobotin ja tietokoneen välillä sarjaportin kautta. PySerial-kirjasto hoitaa sarjaporttiyhteyden tekniset yksityiskohdat ja tarjoaa käyttöliittymän datan lähettämiseen ja vastaanottamiseen laitteiden välillä PyPrinterControl-kirjaston käyttöön. (12.)

3.4 Testausrobotin ohjauslogiikka

3.4.1 Testilaitteen tunnistus testiympäristöstä

Projektin testiautomaatioympäristö on suunniteltu siten, että testikehykseen on tallennettuna tietyt laite- ja sovelluskohtaiset muuttujat, jotka määrittävät muun muassa testaamisessa käytettävän laitteen ja testattavan sovelluksen asetukset. Nämä laite- ja sovelluskohtaiset muuttujat on määritelty testiympäristön setup.robot-tiedostossa (kuva 9).



Kuva 9. Laite- ja sovelluskohtaiset asetukset on määritelty testiympäristön setup.robot-tiedostoon.

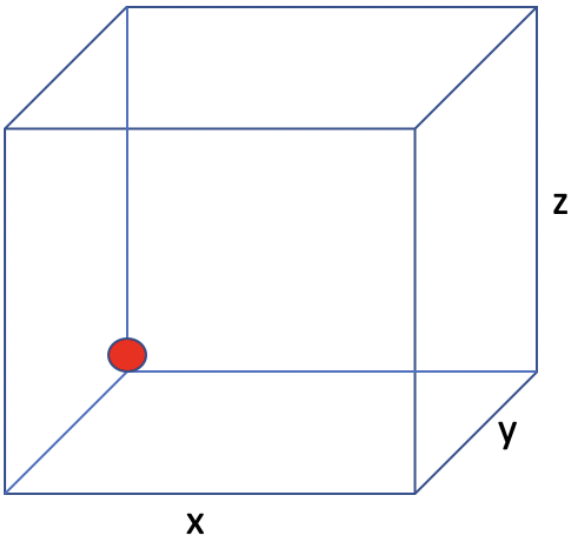
Setup.robot-tiedostossa määritellyt muuttujat ovat tärkeitä jokaisen testitapauksen alussa, sillä ne ohjaavat testirobotin kosketuspäätä siirtymään testilaitteen läheisyyteen ja varmistavat sovelluksen käynnistymisen oikealle laitteelle. Laitekohtaista DEVICE-muuttujaa käytetään fyysisen testausrobotin kirjautumisprosessin suorittamiseen halutulle laitteelle, kun taas DEVICE_ID-muuttujaa hyödynnetään oikean testilaitteen ohjaamiseen Robot Frameworkin AppiumLibrary-kirjaston avulla (kuva 10).

```
Launch Mobile Application
    Move Robot To Start Position    ${DEVICE}    # PyRobotControl.py

    Open Application    ${APPIUM_SERVER}
    ...    deviceName=${DEVICE}
    ...    automationName=${AUTOMATION_NAME}
    ...    platformName=${DEVICE_PLATFORM}
    ...    udid=${DEVICE_ID}
    ...    app=${APP_LOCATION}
    ...    appPackage=${APP_PACKAGE}
    ...    appActivity=${APP_ACTIVITY}
    ...    noReset=${NoReset}
    ...    forceAppLaunch=${ForceAppLaunch}
    ...    shouldTerminateApp=${TerminateApp}
```

Kuva 10. Laitekohtaisten muuttujien avulla Robot Framework tietää, mille testilaitteelle testattava sovellus avataan, ja mitä laitetta testausrobotin pitää ohjata.

3.4.2 Robotin ohjaaminen alustan koordinaattien perusteella

Testausrobotin ohjaaminen testausalustan päällä perustuu alustan koordinaatistoon. Alustan mitat ovat 260 mm pituutta, 260 mm leveyttä ja 220 mm korkeutta. Kosketuspään nollakoordinaatit ovat x- ja y-akseleiden suuntaisesti alustan nurkassa ja korkeussuunnassa alustaa vasten (kuva 11). Robotin kosketuspäätä liikutetaan G-koodikomentojen avulla x-, y- ja z-akseleita pitkin ja sitä liikutetaan testiympäristöstä käsin syöttämällä robotille G-koodikomennot, joissa koordinaatit ilmoitetaan millimetreinä.



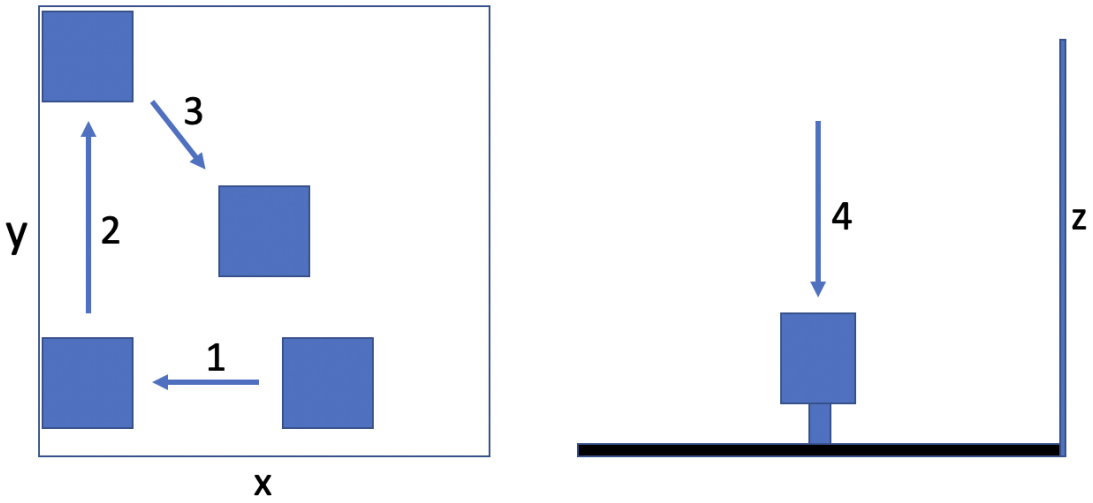
Kuva 11. Robotin kosketuspäätä liikutetaan alustan päällä x-, y- ja z-akselien mukaisesti syöttämällä laitteelle G-koodikomentoja, joka määrittävät, mihin sijaintiin tulostuspää liikkuu. Tulostimen jokaisen akselin nolla-arvo sijaitsee laitteen alanurkassa (merkitty punaisella).

3.4.3 Robotin automaattinen kalibrointi testiajon alussa

Kun testiajo käynnistyy, testausrobotti suorittaa automaattisen kalibroinnin ennen ensimmäisen testitapauksen suorittamista. Tämä kalibrointiprosessi varmistaa, että laitteen kosketuspään sijaintitiedot ovat määritelty oikein. Se estää mahdolliset virheet tai tekniset ongelmat, kuten törmäilyt testilaitteisiin tai alustaan.

Tulostimen kalibrointiprosessi alkaa sillä, että se nostaa laitteen kosketuspäätä 5 mm ylöspäin. Sen jälkeen kosketuspää lähtee liikkumaan kohti alustan reunaa x-akselin suuntaisesti, kunnes se törmää ja tunnistaa reunan. Tämän jälkeen kosketuspää liikkuu y-akselin suuntaisesti kohti alustan reunaa, toistaen saman hallitun törmäämisen. Reunantunnistusliikkeiden avulla robotti pystyy laskemaan alustan vertikaalisen keskipisteen (kuva 12).

Kun kosketuspään sijainti on kalibroitu x- ja y-akselien suuntaisesti, se siirtyy alustan keskelle ja laskeutuu sitä vasten, kunnes se tunnistaa olevansa siinä kiinni (kuva 12). Tämän jälkeen robotti tietää kosketuspäänsä sijainnin kaikkien akselien suuntaisesti eikä liikuta sitä virheellisesti väärään suuntaan missään testausprosessin vaiheessa.



Kuva 12. Testausrobotti kalibroi kosketuspäänsä sijainnin ennen ensimmäistä testitapausta mittaamalla sen sijainnin etsimällä ensin x- ja y-akselien suuntaisesti alustan reunat (kuvassa vaiheet 1 ja 2) ja siirtymällä sitten alustan keskelle, jossa robotti laskeutuu testausalustaa vasten (kuvassa vaiheet 3 ja 4).

Automaattisen kalibroinnin jälkeen robotti siirtää kosketuspäänsä valitun testilaitteen yläpuolella olevaan testilaitekohtaiseen oletussijaintiin. Tämä nopeuttaa testausrobotin suorittamaa kirjautumisprosessia tulevissa testitapauksissa. Kalibrointia ei tarvitse suorittaa uudelleen jokaisen testin yhteydessä, koska robotti muistaa tallennetut sijaintitiedot, kunnes sen virta katkaistaan. Turvallisuussyistä robotti on asetettu suorittamaan automaattisen kalibroinnin aina ennen ensimmäisen testitapauksen suorittamista, vaikka sitä ei koskaan sammutettaisi.

3.4.4 Robotin suorittama kirjautumisvaihe

Kun testausrobotti on kalibroinut itsensä ja siirtänyt kosketuspäänsä laitekohtaiseen oletussijaintiin, se odottaa aktivointikäskyä testiautomaatiokehykseltä. Samalla Robot Framework asentaa testattavan sovelluksen testilaitteeseen ja aloittaa testitapauksen suorittamisen Appiumin verkko-ohjainprotokollaa käyttäen.

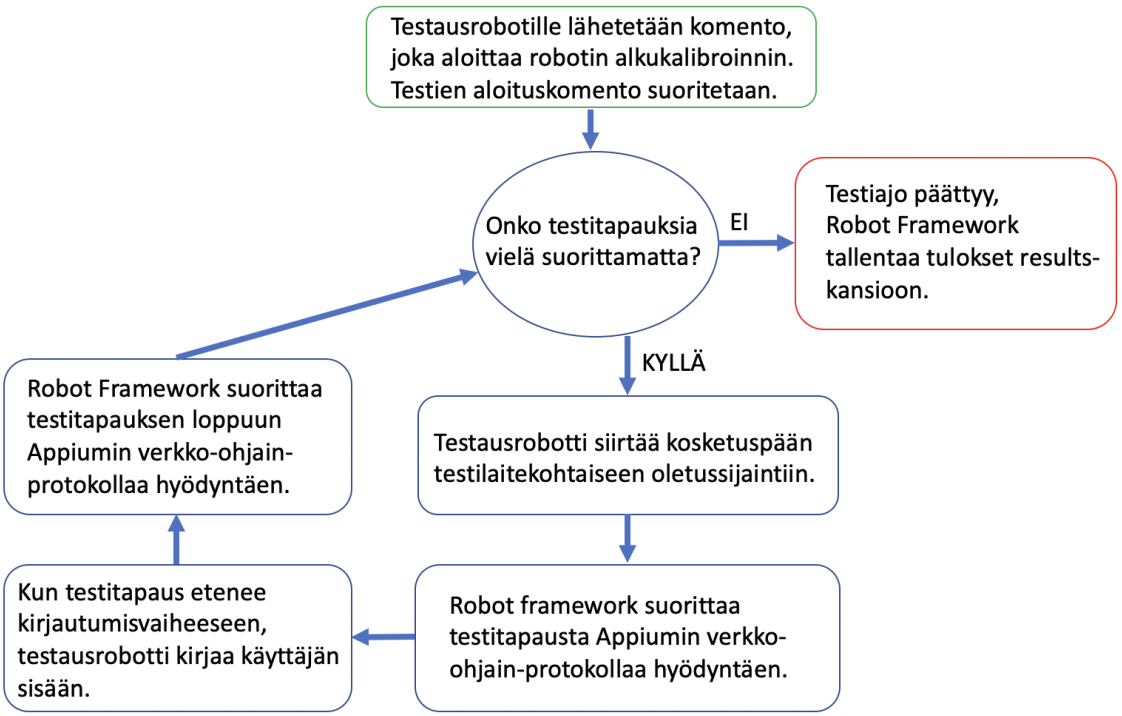
Kun testitapaus etenee kirjautumisvaiheeseen ja PIN-koodin syöttöön, Appiumin käyttö keskeytyy ja fyysinen testausrobotti aktivoituu. Robotti syöttää PIN-koodin sovellukseen ja kirjaa testikäyttäjän sisään käyttämällä laitteen kosketusnäyttöä. Tämän jälkeen kosketuspää palaa takaisin laitekohtaiseen oletussijaintiin odottamaan seuraavaa aktivointikutsua.

Testitapauksen suorittaminen jatkuu tämän jälkeen Appiumin avulla, kunnes testiautomaatiokehys aktivoi testausrobotin uudelleen seuraavan testitapauksen alkaessa.

4 TESTAUSROBOTIN TESTAUS JA VARMENNUS

4.1 Testiautomaatioympäristö

Tässä luvussa esitellään tarkemmin testausrobotin käyttöä testiautomaatioympäristön osana. Robotin testaamista ja varmennusta varten on luotu uusi testiautomaatioympäristö, joka sisältää vain robotin toiminnallisuuden osoittamiseen tarvittavat olennaiset asiat. Kuvassa 13 kuvataan testiajon kulkua yksinkertaisesti ja myöhemmin tässä luvussa käsitellään tarkemmin sen toimintaa.

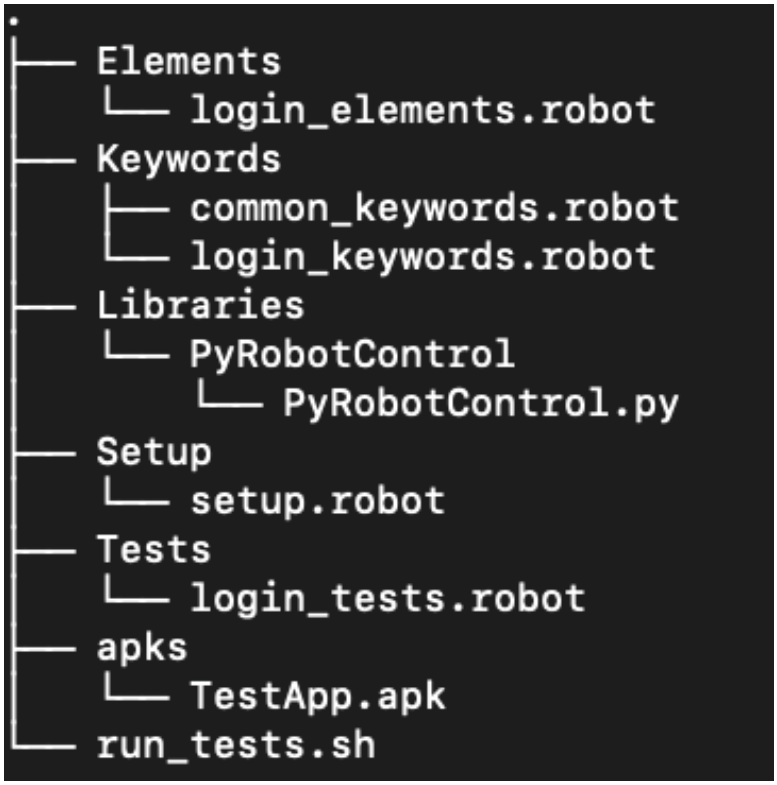


Kuva 13. Testiajon eteneminen.

Testiympäristöön on luotu kaksikymmentä identtistä testitapausta, mikä auttaa varmistamaan testausrobotin toimivuuden peräkkäisten testien aikana.

Alkukalibrointi, joka suoritetaan testiautomaatioprosessin alussa, lisää testauksen kestoa noin kahdellakymmenellä sekunnilla. Kuitenkin, koska kalibrointi suoritetaan vain ensimmäisen testitapauksen yhteydessä, se ei vaikuta merkittävästi usean testitapauksen suorittamisen kokonaiskestoon.

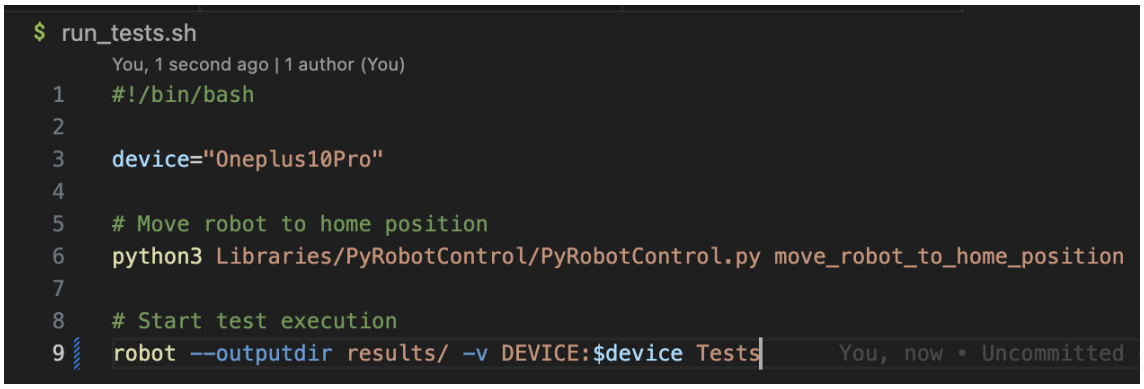
Robot Framework-testiautomaatioympäristön kansiorakenne on esitetty kuvassa 14. Opinnäytetyön liitteet-osiossa on saatavilla kaikki tässä luvussa mainitut testiympäristön tiedostot.



Kuva 14. Robotin toiminnan varmentamiseen käytetyn testiautomaatioympäristön kansiorakenne.

4.2 Testiajon eteneminen

Testiautomaatioprosessi käynnistetään suorittamalla testiympäristöön lisätty run_tests.sh-tiedosto (kuva 15). Tiedoston rivillä kolme määritellään mobiililaite, jota käytetään testauksessa, ja rivillä viisi Python-komento kutsuu PyRobotControl-kirjaston (liitteet 1, 2 ja 3) move_robot_to_home_position()-funktiota. Tämä funktio lähettää testirobotille käskyn suorittaa alkukalibrointi ja siirtää robotin kosketuspää testausalustan keskelle odottamaan aktivointikäskyä testiympäristöltä. Tiedoston rivillä 9 annetaan robot-komento, joka käynnistää login_tests.robot-tiedostossa (liite 4) sijaitsevien testitapausten suorittamisen.



Kuva 15. run_tests.sh-tiedoston suorittaminen aloittaa testiajon.

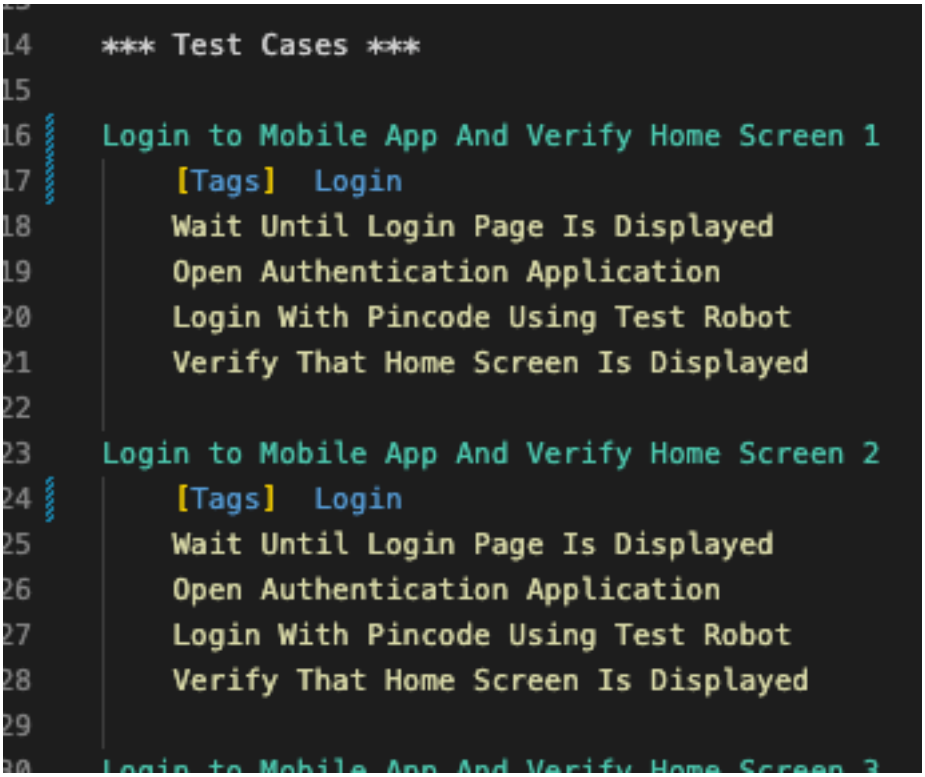
Kun robotin alkukalibrointi on suoritettu, testitapausten suorittaminen alkaa automaattisesti. Jokaisen testitapauksen alussa Robot Framework suorittaa common_keywords.robot-tiedostossa (liite 5) määritellyn avainsanan Launch Mobile Application, joka siirtää robotin kosketuspään testilaitteen läheisyyteen ja käynnistää testattavan sovelluksen (kuva 16).

```
Launch Mobile Application
    Move Robot To Start Position   ${DEVICE}    # PyRobotControl.py

    Open Application    ${APPIUM_SERVER}
        ...    deviceName=${DEVICE}
        ...    automationName=${AUTOMATION_NAME}
        ...    platformName=${DEVICE_PLATFORM}
        ...    udid=${DEVICE_ID}
        ...    app=${APP_LOCATION}
        ...    appPackage=${APP_PACKAGE}
        ...    appActivity=${APP_ACTIVITY}
        ...    noReset=${NoReset}
        ...    forceAppLaunch=${ForceAppLaunch}
        ...    shouldTerminateApp=${TerminateApp}
```

Kuva 16. Launch Mobile Application -avainsana suoritetaan jokaisen testitapauksen alussa. Se sisältää kaksi avainsanaa: Avainsana Move Robot To Start Position kutsuu PyRobotControl-kirjaston move_robot_to_start_position()-funktiota, joka siirtää robotin kosketuspään valitun testilaitteen läheisyyteen ja avainsana Open Application avaa testattavan sovelluksen valitulla testilaitteella.

Tässä projektissa määritellyt identtiset testitapaukset (Kuva 17) etenevät seuraavalla tavalla: Odotetaan, että sovellus on avautunut testilaitteella ja sen etusivu on näkyvillä. Tämän jälkeen avataan tunnistussovellus, testausrobotti kirjaa testikäyttäjän sisään ja lopuksi vahvistetaan kirjautumisen onnistuminen tarkistamalla, että testisovelluksen etusivulla oleva "welcome"-teksti on näkyvillä.

Testitapausten käyttämät avainsanat löytyvät testiympäristön login_keywords.robot-tiedostosta (liite 6).



Kuva 17. Testiympäristöön on luotu kaksikymmentä identtistä testitapausta havainnollistamaan robotin käytön vaikutusta testiajon kokonaiskestoon.

Kun kaikki testitapaukset on suoritettu, tulokset tallentuvat automaattisesti testiympäristön "results"-kansioon, josta niitä voi tarkastella.

4.3 Testiajon toistaminen testausrobotin toiminnan varmistamiseksi

Suoritin kaksikymmentä identtistä testitapausta satunnaisesti neljällä eri mobiililaitteella varmistaakseni testausrobotin luotettavuuden, ja tulokset olivat vakuuttavia. Yksikään testitapaus ei keskeytynyt kirjautumisvaiheessa. Robotin alkukalibrointi suoritettiin ennen varsinaista testiajoa, joten se ei näy Robot Frameworkin luomassa raportissa (kuva 18).

Testi suoritettiin tietyn yrityksen mobiilisovellusta vasten, ja projektin yleisluonteisuuden vuoksi päätin jättää tarkemmat raportin lokitiedot ja kuvat sovelluksen testaamisesta pois.

Test Details LOG

All | Tags | Suites | Search

Tag: Login

Status: 19 tests total, 19 passed, 0 failed, 0 skipped

Total Time: 00:07:55.621

Name	Status	Elapsed	Start / End
Tests . Login Tests . Login to Mobile App And Verify Home Screen 1	PASS	00:00:24.952	20230510 17:41:21.465 20230510 17:41:46.417
Tests . Login Tests . Login to Mobile App And Verify Home Screen 10	PASS	00:00:24.862	20230510 17:45:06.875 20230510 17:45:31.737
Tests . Login Tests . Login to Mobile App And Verify Home Screen 11	PASS	00:00:24.906	20230510 17:45:31.737 20230510 17:45:56.643
Tests . Login Tests . Login to Mobile App And Verify Home Screen 13	PASS	00:00:25.145	20230510 17:45:56.643 20230510 17:46:21.788
Tests . Login Tests . Login to Mobile App And Verify Home Screen 14	PASS	00:00:24.903	20230510 17:46:21.788 20230510 17:46:46.691
Tests . Login Tests . Login to Mobile App And Verify Home Screen 15	PASS	00:00:24.669	20230510 17:46:46.691 20230510 17:47:11.360
Tests . Login Tests . Login to Mobile App And Verify Home Screen 16	PASS	00:00:25.003	20230510 17:47:11.360 20230510 17:47:36.363
Tests . Login Tests . Login to Mobile App And Verify Home Screen 17	PASS	00:00:24.678	20230510 17:47:36.363 20230510 17:48:01.041
Tests . Login Tests . Login to Mobile App And Verify Home Screen 18	PASS	00:00:24.433	20230510 17:48:01.041 20230510 17:48:25.474
Tests . Login Tests . Login to Mobile App And Verify Home Screen 19	PASS	00:00:25.037	20230510 17:48:25.474 20230510 17:48:50.511
Tests . Login Tests . Login to Mobile App And Verify Home Screen 2	PASS	00:00:24.677	20230510 17:41:46.417 20230510 17:42:11.094
Tests . Login Tests . Login to Mobile App And Verify Home Screen 20	PASS	00:00:26.577	20230510 17:48:50.511 20230510 17:49:17.088
Tests . Login Tests . Login to Mobile App And Verify Home Screen 3	PASS	00:00:24.982	20230510 17:42:11.094 20230510 17:42:36.076
Tests . Login Tests . Login to Mobile App And Verify Home Screen 4	PASS	00:00:24.889	20230510 17:42:36.076 20230510 17:43:00.965
Tests . Login Tests . Login to Mobile App And Verify Home Screen 5	PASS	00:00:25.125	20230510 17:43:00.965 20230510 17:43:26.090
Tests . Login Tests . Login to Mobile App And Verify Home Screen 6	PASS	00:00:24.664	20230510 17:43:26.091 20230510 17:43:50.755
Tests . Login Tests . Login to Mobile App And Verify Home Screen 7	PASS	00:00:25.946	20230510 17:43:50.755 20230510 17:44:16.701
Tests . Login Tests . Login to Mobile App And Verify Home Screen 8	PASS	00:00:23.718	20230510 17:44:16.701 20230510 17:44:40.419
Tests . Login Tests . Login to Mobile App And Verify Home Screen 9	PASS	00:00:26.455	20230510 17:44:40.420 20230510 17:45:06.875

Kuva 18. reports.html. Yksikään testitapaus ei pysähtynyt kirjautumisvaiheeseen.

5 TULOKSET JA JATKOKEHITYS

5.1 Tulokset

Fyysisen testausrobotin valmistaminen 3D-tulostinta muokkaamalla ei ollut monimutkainen prosessi. Tulostimen muuttaminen robotiksi vaati vain pieniä laitteeseen tehtäviä fyysisiä muutoksia. Tähän projektiin valittu Ender 7 -3D-tulostin osoittautui hyväksi valinnaksi, sillä sen liittäminen testiautomaatioympäristöön oli mutkatonta ja onnistui ongelmitta. Opinnäytetyötä tehdessä muiden tulostinmallien kokeilu ei ollut mahdollista, joten suorituskykyvertailua laitteiden välillä ei voitu suorittaa.

Muokatun tulostimen ohjelmistoon ei tarvinnut tehdä muutoksia lainkaan ja sitä ohjaavalle tietokoneelle ei tarvinnut asentaa ylimääräisiä työkaluja tai ohjelmistoja. Testausrobotin ohjaamisen mahdollistamien Robot Framework -testiautomaatioympäristöstä onnistui vakiintuneita laiteohjausmetodeja käyttäen ja joustavan testikehyksen ansiosta se toimi saumattomasti ohjelmallisen automaatiotestauksen tukena.

Luvussa 4.3 robotin käyttöä testattiin toistamalla kaksikymmentä identtistä testitapausta satunnaisesti ympäristöön lisätyillä neljällä mobiililaitteella ja robotin toiminta oli varmaa ja sen käyttö osoittautui luotettavaksi. Testausrobotti toimi määrätyllä tavalla osana mobiilitestausympäristöä ja varmennustestaus osoitti laitteen luotettavuuden. Tämän opinnäytetyön valmistuessa robotti on aktiivisessa käytössä osana yrityksen testiautomaatioympäristöä, jota varten se kehitettiin.

5.2 Jatkokehitys

Jatkossa robottia voisi kehittää niin, että se hyödyntäisi kameraa ja koneoppimista testaamisen parantamiseksi. Tässä opinnäytetyössä esitelty testausrobotti navigoi vain valmiiksi ohjelmoituja reittejä pitkin, eikä varsinaisesti lue testilaitteiden tiloja tai kommunikoi niiden kanssa. Tämä tarkoittaa sitä, että robotin käyttömahdollisuudet ovat hyvin rajoittuneita ja se pystyy tällä hetkellä toistamaan vain tarkkaan määritetyt toiminnot.

Robottia voisi kehittää älykkäämmäksi ja monipuolisemmaksi testaustyökaluksi lisäämällä siihen kameran, joka ottaa kuvia laitteiden näytöistä ja testattavasta sovelluksesta. Se voisi käyttää kuvia esimerkiksi sovelluksessa navigoimiseen ja laitteen paikantamiseen riippumatta sen sijainnista testausalustalla.

Koneoppimista hyödyntäen robotti voisi oppia tunnistamaan kuvista sovelluksen sisältöä ja parantamaan testauksen laatua. Se mahdollistaisi minkä tahansa mobiililaitteen käyttämisen testilaitteena ja koko sovelluksen testaamisen ihmiskäyttäjän tavoin, koska silloin robotti osaisi navigoida testattavassa sovelluksessa itsenäisesti kosketusnäytön kautta. Koneoppimisalgoritmeja käyttämällä robotti voi myös oppia aiemmista testituloksista ja parantaa tarkkuuttaan ajan myötä.

6 POHDINTA

Opinnäytetyön tavoitteena oli kuvata fyysisen testausrobotin suunnittelua ja toteuttamista ohjelmallisen automaation tueksi. Robotille oli tarve yrityksessä, joka tarvitsi fyysisen ratkaisun mobiilitestiautomaatioprosessien tukemiseen. Vaikka tarve tuli tältä tietyltä yritykseltä, ratkaisun yleisluonteisuus mahdollistaa tämän opinnäytetyön hyödyntämisen myös muissa vastaavanlaisissa testiautomaatioratkaisuissa. Siksi opinnäytetyö päätettiin toteuttaa niin, että se kuvaa testausrobotin käyttöä yleisesti testiautomaation tukena minkä tahansa sovelluksen testaamiseen.

Työn tekeminen oli kokonaisuudessaan erittäin mielenkiintoinen prosessi, koska se syntyi aidosta tarpeesta ja yhdisti testiautomaation ja robotiikan. Ongelmaan ei ollut suoraa ratkaisua valmiina ja sain suunnitella ja toteuttaa projektin itse. Aihe kiinnosti minua myös henkilökohtaisesti, sillä tässä työssä käytetyt testiautomaatiotyökalut olivat minulle tuttuja.

Minulla ei ollut aiempaa kokemusta robottien rakentamisesta tai työskentelystä robotiikan parissa. Laiteohjauksen ja 3D-tulostimen toiminnasta oppiminen oli kiinnostavaa ja oli hienoa päästä kokeilemaan jotain uutta. Tämä projekti tarjosi minulle mahdollisuuden laajentaa osaamistani ja oppia uusia taitoja.

En tiennyt, mitä odottaa tältä projektilta, sillä minulla ei ollut aikaisempaa kokemusta 3D-tulostinten ohjaamisesta, saati fyysisten robottien käyttämisestä Robot Framework -testiautomaatiokehyksen rinnalla. Robotin suunnittelu ja rakentaminen sujui kuitenkin ilman suurempia ongelmia ja olen lopputulokseen hyvin tyytyväinen.

Työn tarkoitus oli suunnitella ja toteuttaa fyysinen testausrobotti mobiiliautomaatiotestaukseen tueksi ja mielestäni opinnäytetyön tuottamien yleisluontoisena toimi siinä hyvin. Työstä jouduttiin jättämään pois joitain yksityiskohtia, kuten tarkempia kuvia robotin käytöstä testattavan sovelluksen kanssa ja yksityiskohtaisemmat lokitiedot testien suorituksesta. Kuvat sovelluksen testaamisesta ja tarkemmista lokitiedoista olisivat ehkä tarjonneet tarkemman kuvan robotin toiminnasta, mutta mielestäni oli parempi ratkaisu pitää opinnäytetyö yleisluontoisena ja jättää sovellukset tunnistamattomiksi.

LÄHTEET

1. Fewster, Mark & Graham, Dorothy 1999. Software Test Automation: Effective use of test execution tools. Harlow: Addison-Wesley Publishing Co.

2. Robot Framework. Robot framework introduction. Robot Framework ry. Hakupäivä 10.5.2023 https://robotframework.org/

3. ISO/IEC IEEE 29119-1 2021, Software and systems engineering — Software testing — Part 1: Concepts and definitions. Hakupäivä 10.5.2023 https://www.iso.org/obp/ui#iso:std:iso-iec-ieee:29119:-1:dis:ed-2:v1:en

4. Appium 2023. Overview of Appium. Hakupäivä 10.5.2023. https://appium.io/docs/en/2.0/intro/

5. Laihonen, Paul 2021. Robot Framework: Past, Present and Future. Eficode.com 20.6.2014. Hakupäivä 10.5.2023. https://www.eficode.com/blog/en/blog/robot-framework

6. Robot Framework. Robot Framework Foundation 2023. Robot Framework ry. Hakupäivä 10.5.2023. https://robotframework.org/foundation/

7. Robot Framework documentation 2023. Selenium Library. Robot Framework ry. Hakupäivä 10.5.2023. https://robotframework.org/SeleniumLibrary/SeleniumLibrary.html

8. Robot Framework documentation 2023. Appium Library. Robot Framework ry. Hakupäivä 10.5.2012. https://serhatbolsu.github.io/robotframework-appiumlibrary/AppiumLibrary.html

9. Robot Framework 2023. Robot Framework User Guide: 1.1.2 High-level architecture. Robot Framework ry. Hakupäivä 10.5.2023 https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html

10. Verma, Nishant 2017. Mobile Test Automation with Appium: Comprehensive guide to build mobile test automation solution using Appium. Birmingham: Packt Publishing Ltd. Hakupäivä 10.5.2023. Packt Publishing eBooks. Vaatii käyttöoikeuden.

https://books.google.fi/books?hl=fi&lr=&id=BHg5DwAAQBAJ&oi=fnd&pg=PP1&dq=mobile+test+automation&ots=AkInlodK5_&sig=_sVm9WsO3UYjLVTNzNbGArxrrg0&redir_esc=y#v=onepage&q=mobile%20test%20automation&f=false

11. Overby, Alan 2011. CNC Machining Handbook. The McGraw-Hill Companies, Inc. Hakupäivä 10.5.2023. O'Reilly. Vaatii käyttöoikeuden. https://learning.oreilly.com/library/view/cnc-machining-handbook/9780071623025/

12. pySerial. Overview. Hakupäivä 10.5.2023. https://pyserial.readthedocs.io/en/latest/pyserial.html,

LIITTEET

PyRobotControl.py, osa 1 liite 1

PyRobotControl.py, osa 2 liite 2

PyRobotControl.py, osa 3 iite 3

login_tests.robot liite 4

common_keywords.robot liite 5

login_keywords.robot liite 6

login_elements.robot liite 7

setup.robot liite 8

run_tests.sh liite 8

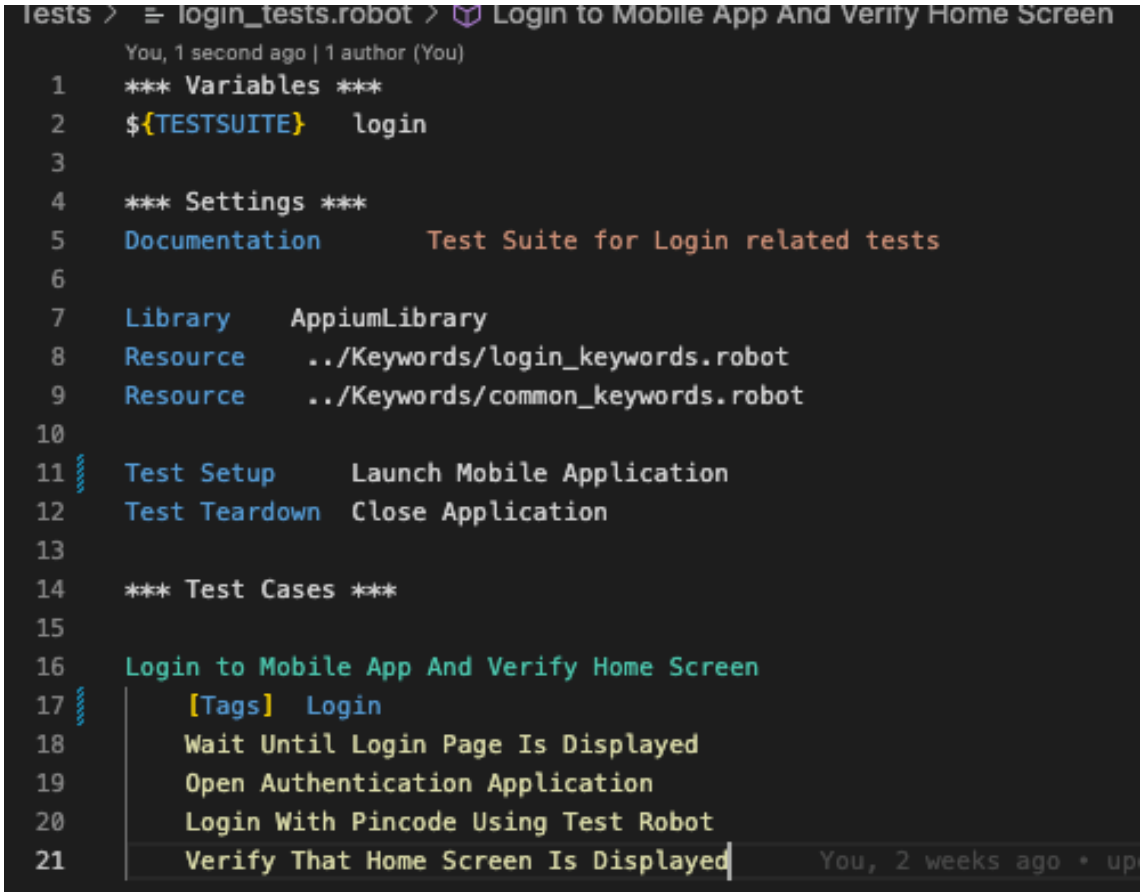
Libraries > PyRobotControl > PyRobotControl.py > ...

You, 12 minutes ago | 1 author (You)

```python
import argparse
import serial
import time

class Movement:
    def __init__(self, port):
        # Initialize serial connection to robot
        self.ser = serial.Serial(port, 115200)


    def __del__(self):
        # Check if serial connection exists before closing
        if hasattr(self, 'ser'):
            # Close serial connection when object is destroyed
            self.ser.close()

    def x_axis(self, pos):
        time.sleep(1)
        # Send G-code command to move to the specified X coordinate
        command = f'G0 X{pos}\n'
        self.ser.write(command.encode())

    def y_axis(self, pos):
        time.sleep(1)
        # Send G-code command to move to the specified Y coordinate
        command = f'G0 Y{pos}\n'
        self.ser.write(command.encode())

    def z_axis(self, pos):
        time.sleep(1)
        # Send G-code command to move to the specified Z coordinate
        command = f'G0 Z{pos}\n'
        self.ser.write(command.encode())

    def home(self):
        # Go to robots home position
        time.sleep(1)
        self.ser.write(b'M106 S0\n')
        time.sleep(1)
        self.ser.write(b'G0 Z40.00\n')
        time.sleep(1)
        self.ser.write(b'G28\n')
        self.ser.write(b'G0 Z25.00\n')

    def start_position(self, device):
        # Go to test device specific default position
        if device == "Oneplus10Pro":            # Oneplus 10 Pro
            self.ser.write(b'G0 X30.00\n')
            self.ser.write(b'G0 Y45.00\n')
        elif device == "SamsungGalaxyS21":      # Samsung Galaxy S21
            self.ser.write(b'G0 X213.00\n')
            self.ser.write(b'G0 Y35.00\n')
        elif device == "SamsungGalaxyA53":      # Samsung Galaxy A53
            self.ser.write(b'G0 X225.00\n')
            self.ser.write(b'G0 Y218.00\n')
        elif device == "GooglePixel6":          # Google Pixel 6
            self.ser.write(b'G0 X45.00\n')
            self.ser.write(b'G0 Y230.00\n')
        else:
            raise ValueError("Invalid device name: {}".format(device))

```

```
62
63  if __name__ == '__main__':
64      parser = argparse.ArgumentParser()
65      parser.add_argument('function', choices=['move_robot_to_home_position'])
66      args = parser.parse_args()
67
68      if args.function == 'move_robot_to_home_position':
69          m = Movement('/dev/tty.usbserial-21310')
70          m.home()
71
72  def move_robot_to_home_position():
73      m = Movement('/dev/tty.usbserial-21310')
74      m.home()
75
76  def move_robot_to_start_position(device):
77      m = Movement('/dev/tty.usbserial-21310')
78      m.start_position(device)
79
80  def login_with_pincode_by_button_presses_(device):
81      m = Movement('/dev/tty.usbserial-21310')
82      if device == "Oneplus10Pro":        # Oneplus 10 Pro
83          m.z_axis(25)
84          m.x_axis(30)
85          m.y_axis(45)
86          m.z_axis(17)
87          m.z_axis(25)
88          m.y_axis(60)
89          m.z_axis(17)
90          m.z_axis(25)
91          m.y_axis(45)
92          m.z_axis(17)
93          m.z_axis(25)
94          m.y_axis(60)
95          m.z_axis(17)
96          m.z_axis(25)
97          m.y_axis(45)
98          m.z_axis(17)
99          m.z_axis(25)
100         m.y_axis(60)
101         m.z_axis(17)
102         m.z_axis(25)
103         m.start_position(device)
```

```
    elif device == "SamsungGalaxyS21":     # Samsung Galaxy S21
        m.z_axis(25)
        m.x_axis(213)
        m.y_axis(35)
        m.z_axis(16)
        m.z_axis(25)
        m.x_axis(197)
        m.z_axis(16)
        m.z_axis(25)
        m.x_axis(213)
        m.z_axis(16)
        m.z_axis(25)
        m.x_axis(197)
        m.z_axis(16)
        m.z_axis(25)
        m.x_axis(213)
        m.z_axis(16)
        m.z_axis(25)
        m.x_axis(197)
        m.z_axis(16)
        m.z_axis(25)
        m.start_position(device)
    elif device == "SamsungGalaxyA53":       # Samsung Galaxy A53
        m.z_axis(25)
        m.x_axis(225)
        m.y_axis(218)
        m.z_axis(15)
        m.z_axis(25)
        m.y_axis(203)
        m.z_axis(15)
        m.z_axis(25)
        m.y_axis(218)
        m.z_axis(15)
        m.z_axis(25)
        m.y_axis(203)
        m.z_axis(15)
        m.z_axis(25)
        m.y_axis(218)
        m.z_axis(15)
        m.z_axis(25)
        m.y_axis(203)
        m.z_axis(15)
        m.z_axis(25)
        m.start_position(device)
    elif device == "1A151FDF600F23":    # Google Pixel 6
        m.z_axis(25)
        m.x_axis(45)
        m.y_axis(230)
        m.z_axis(15)
        m.z_axis(25)
        m.x_axis(57)
        m.z_axis(15)
        m.z_axis(25)
        m.x_axis(45)
        m.z_axis(15)
        m.z_axis(25)
        m.x_axis(57)
        m.z_axis(15)
        m.z_axis(25)
        m.x_axis(45)
        m.z_axis(15)
        m.z_axis(25)
        m.x_axis(57)
        m.z_axis(15)
        m.z_axis(25)
        m.start_position(device)
    else:
        raise ValueError("Invalid device name: {}".format(device))
```


Tests > login_tests.robot > Login to Mobile App And Verify Home Screen
You, 1 second ago | 1 author (You)
*** Variables ***
${TESTSUITE} login
*** Settings ***
Documentation Test Suite for Login related tests
Library AppiumLibrary
Resource ../Keywords/login_keywords.robot
Resource ../Keywords/common_keywords.robot
Test Setup Launch Mobile Application
Test Teardown Close Application
*** Test Cases ***
Login to Mobile App And Verify Home Screen
[Tags] Login
Wait Until Login Page Is Displayed
Open Authentication Application
Login With Pincode Using Test Robot
Verify That Home Screen Is Displayed
You, 2 weeks ago • up

Keywords > common_keywords.robot > ...

You, 1 second ago | 1 author (You)

```
*** Settings ***
Library     AppiumLibrary
Library     BuiltIn
Library     ../Libraries/PyRobotControl/PyRobotControl.py
Resource    ../Setup/setup.robot

*** Keywords ***

Launch Mobile Application
    Move Robot To Start Position   ${DEVICE}    # PyRobotControl.py

    Open Application    ${APPIUM_SERVER}
        ...    deviceName=${DEVICE}
        ...    automationName=${AUTOMATION_NAME}
        ...    platformName=${DEVICE_PLATFORM}
        ...    udid=${DEVICE_ID}
        ...    app=${APP_LOCATION}
        ...    appPackage=${APP_PACKAGE}
        ...    appActivity=${APP_ACTIVITY}
        ...    noReset=${NoReset}
        ...    forceAppLaunch=${ForceAppLaunch}
        ...    shouldTerminateApp=${TerminateApp}
```

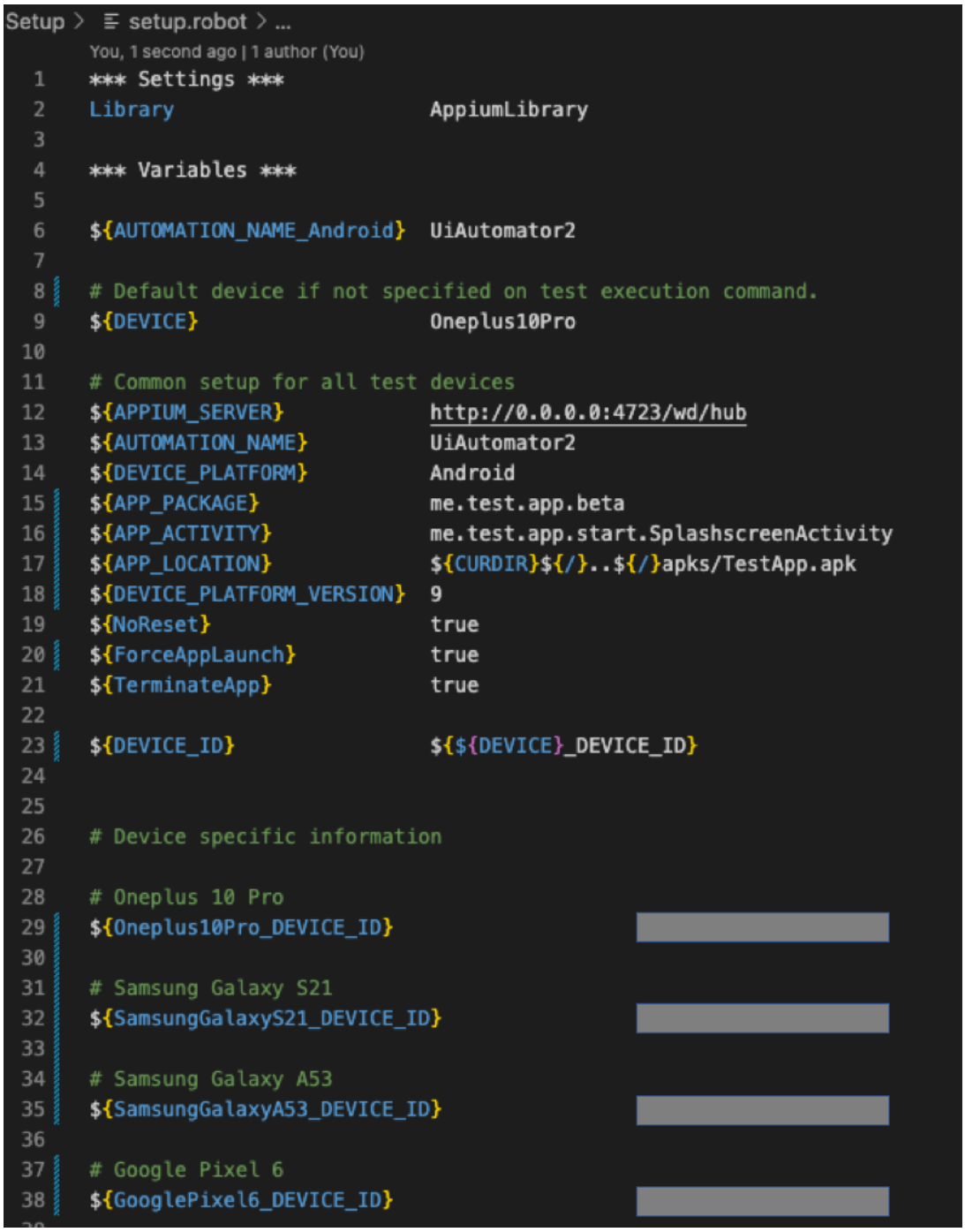
Keywords > login_keywords.robot > ...

```
1   *** Settings ***
2   Library     ../Libraries/PyRobotControl/PyRobotControl.py
3   Resource    ../Elements/login_elements.robot
4
5   *** Keywords ***
6   Wait Until Login Page Is Displayed
7       Wait Until Element Is Visible           ${LOGIN_WITH_AUTHENTICATOR_BUTTON}
8
9   Open Authentication Application
10      Click Element                           ${LOGIN_WITH_AUTHENTICATOR_BUTTON}
11      Wait Until Element Is Visible           ${OPEN_NUMPAD_BUTTON}    timeout=15s
12      Click Element                           ${OPEN_NUMPAD_BUTTON}
13      Wait Until Element Is Visible           ${NUMPAD}
14
15  Login With Pincode Using Test Robot
16      # PyRobotControl.py
17      Login With Pincode By Button Presses    ${DEVICE}
18
19  Verify That Home Screen Is Displayed
20      Wait Until Element Is Visible           ${HOME_SCREEN_HEADER}    timeout=60s
```

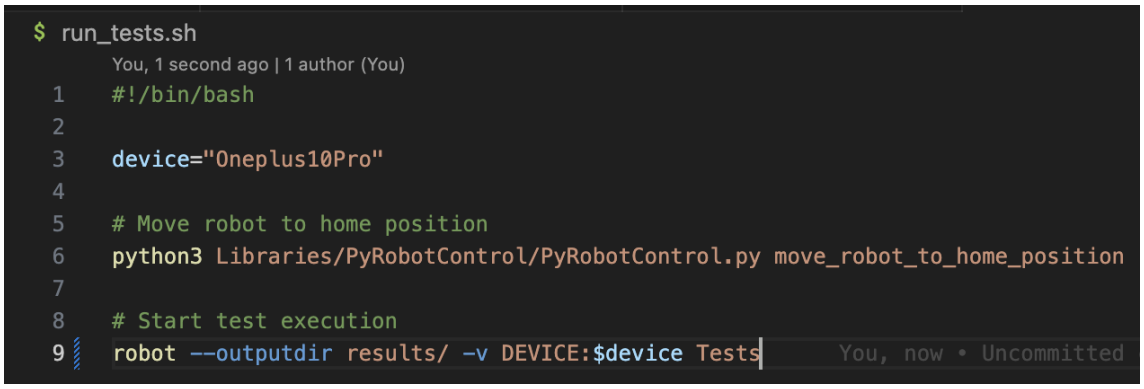
Elements > login_elements.robot > ...

You, 1 second ago | 1 author (You)

```
*** Variables ***

${LOGIN_WITH_AUTHENTICATOR_BUTTON}      xpath=//*[@text="Login with authenticator"]
${OPEN_NUMPAD_BUTTON}                   xpath=//*[@text="Identify with pincode"]
${NUMPAD}                               xpath=//*[@resource-id="me.test.app:id/NumpadLayout"]
${HOME_SCREEN_HEADER_TEXT}              xpath=//*[@text="Welcome!"]
```



```
Setup > setup.robot > ...
      You, 1 second ago | 1 author (You)
1     *** Settings ***
2     Library                       AppiumLibrary
3
4     *** Variables ***
5
6     ${AUTOMATION_NAME_Android}    UiAutomator2
7
8     # Default device if not specified on test execution command.
9     ${DEVICE}                     Oneplus10Pro
10
11    # Common setup for all test devices
12    ${APPIUM_SERVER}              http://0.0.0.0:4723/wd/hub
13    ${AUTOMATION_NAME}            UiAutomator2
14    ${DEVICE_PLATFORM}            Android
15    ${APP_PACKAGE}                me.test.app.beta
16    ${APP_ACTIVITY}               me.test.app.start.SplashscreenActivity
17    ${APP_LOCATION}               ${CURDIR}${/}..${/}apks/TestApp.apk
18    ${DEVICE_PLATFORM_VERSION}    9
19    ${NoReset}                    true
20    ${ForceAppLaunch}             true
21    ${TerminateApp}               true
22
23    ${DEVICE_ID}                  ${${DEVICE}_DEVICE_ID}
24
25
26    # Device specific information
27
28    # Oneplus 10 Pro
29    ${Oneplus10Pro_DEVICE_ID}                    [redacted]
30
31    # Samsung Galaxy S21
32    ${SamsungGalaxyS21_DEVICE_ID}                [redacted]
33
34    # Samsung Galaxy A53
35    ${SamsungGalaxyA53_DEVICE_ID}                [redacted]
36
37    # Google Pixel 6
38    ${GooglePixel6_DEVICE_ID}                    [redacted]
```


$ run_tests.sh
You, 1 second ago | 1 author (You)
#!/bin/bash
device="Oneplus10Pro"
# Move robot to home position
python3 Libraries/PyRobotControl/PyRobotControl.py move_robot_to_home_position
Start test execution
robot --outputdir results/ -v DEVICE:$device Tests
You, now • Uncommitted