OULUN AMMATTIKORKEAKOULU

SHAKKIROBOTIN KEHITTÄMINEN YHTEISTYÖROBOTILLA

Juuso Korpinen
Opinnäytetyö (AMK)
Kevät 2026
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma
Ohjelmistokehityksen suuntautumisvaihtoehto

Tekijä: Juuso Korpinen
Opinnäytetyön otsikko: Shakkirobotin kehittäminen yhteistyörobotilla
Työn valmistumislukukausi ja -vuosi: Kevät 2026
Sivumäärä: 31

Opinnäytetyön tavoitteena oli kehittää shakkia pelaava robotti hyödyntämällä Universal Robots UR16e-yhteistyörobottia sekä konenäköteknologiaa. Projektin päämääränä oli luoda kokonaisuus, joka kykenee pelaamaan shakkipelin alusta loppuun saakka tunnistamalla ihmisvastustajan tekemät siirrot fyysisellä shakkilaudalla, analysoimalla pelitilanteen ja suorittamalla robotin vastasiirrot.

Ohjelmisto toteutettiin Python-ohjelmointikielellä. Shakkipelin sääntöjen valvonnasta vastasi python-chess-kirjasto ja pelisiirtojen laskemiseen hyödynnetään Stockfish-shakkimoottoria. Kommunikaatio robotin ja tietokoneen välillä toteutettiin reaaliaikaisen RTDE-rajapinnan kautta ur_rtde-kirjastoa käyttämällä.

Ihmisen tekemien siirtojen havaitsemiseen käytettiin aluksi OpenCV-kuvankäsittelykirjastoa (Bradski 2000), jonka avulla siirrot tunnistettiin vertaamalla kahden kuvan välisiä eroja. Tämä menetelmä osoittautui liian herkäksi ympäristön valaistuksen aiheuttamille varjoille. Ongelman ratkaisemiseksi päätettiin kouluttaa oma YOLO-konenäkömalli, joka opetettiin tunnistamaan shakkinappulat ja niiden sijainnit kuvan avulla. Konenäkömallilla havaittaessa valaistuksen ja varjojen vaikutus pieneni huomattavasti.

Projektin lopputuloksena oli toimiva shakkirobotti, joka pystyy pelaamaan kokonaisen shakkipelin ihmistä vastaan. Lopputulosta on mahdollista jatkokehittää uusilla ominaisuuksilla, kuten eri shakkivarianttien tukemisella.

ABSTRACT

Oulu University of Applied Sciences
Degree Program in Information technology
Option of Software development

Author: Juuso Korpinen
Title of thesis: Developing a chess robot with a collaborative robot
Term and year when the thesis was submitted: Spring 2026
Number of pages: 31

The objective of this thesis was to develop a chess-playing robot by utilizing a Universal Robots UR16e collaborative robot and computer vision technology. The primary goal was to create a robot capable of playing a chess game by detecting the human opponent's moves on a physical chessboard, analyzing the game state, and executing the robot's countermoves.

The software was implemented using the Python programming language. The python-chess library manages the game rules, while the Stockfish engine is utilized to calculate the optimal moves. Communication between the robot and the computer was established via a real-time RTDE interface using the ur_rtde library.

To detect the human player's moves, the OpenCV image processing library was initially used to determine the move by recognizing the difference between two images. This method proved to be too sensitive to the shadows caused by the environment's lighting conditions. To resolve this issue, a custom YOLO computer vision model was trained to recognize the chess pieces and their locations on the board using the camera. Using the computer vision model reduced the interference caused by lighting and shadows.

The outcome of the project was a functional chess-playing robot that could play a full game against a human opponent. In the future, the project can be further developed with new features, such as support for different chess variants.

SISÄLLYS

TIIVISTELMÄ 2
ABSTRACT 3
SISÄLLYS 4
1 JOHDANTO 5
2 TEKNOLOGIAT 6
2.1 Robotin malli 6
2.2 Robotiq-lisäosat 8
2.3 Robotin kanssa kommunikointi 9
2.4 Robotiq-lisäosien kanssa kommunikointi 10
2.5 Shakkimoottori 11
3 ENSIMMÄISEN VERSION TOTEUTUS 12
3.1 Nappuloiden liikuttaminen 12
3.2 Shakkilogiikka 14
3.3 Ensimmäinen konenäkö 15
4 KOULUTETTUA KONENÄKÖMALLIA KÄYTTÄVÄ TOTEUTUS 20
4.1 Uudet nappulat 20
4.2 Konenäkömallin koulutus 22
4.3 Konenäkömallin integrointi 27
5 POHDINTA 29
LÄHTEET 30

1 JOHDANTO

Tämän opinnäytetyön aiheena on shakkirobotin kehittäminen hyödyntämällä UR16e-yhteistyörobottia ja konenäköä. Projektissa käytetään Python-ohjelmointikieltä, jonka käytännön soveltamista ja ohjelmointia harjoitellaan työn aikana.

Aiheeksi valittiin yhteistyörobotin käyttö, sillä Oulun ammattikorkeakoulun Linnanmaan kampuksella oli esitelty useita eri robotteja, joita opiskelijat saisivat käyttää. Käsivarsirobotti valittiin sen monipuolisuuden ja joustavan saatavuuden takia. Muiden robottien käyttöön olisi pitänyt aina varata erillinen aika, jotta niitä pääsisi käyttämään.

Tavoitteena on saada robotti pelaamaan shakkipeli ihmistä vastaan alusta loppuun. Robotin pelaamaksi peliksi valittiin shakki, koska se on selvien sääntöjen ja ruutupohjaisen laudan takia erinomainen vaihtoehto. Monimutkaisempien pelien logiikka ja konenäkö olisi saattanut koettua liian haasteellisiksi toteuttaa projektin aikarajassa.

Työ toteutetaan Python-ohjelmointikielellä, sillä sitä suosittelivat robottia esitelleet opiskelijat, jotka olivat aikaisemmin hyödyntäneet sitä robotiikkaprojektissa. Projekti on myös mahdollista toteuttaa C++-ohjelmointikielellä, mutta Python valittiin sen helppokäyttöisyyden ja tuttavuuden takia. Projektin versionhallinta hoidetaan GitHubilla.

Ennen projektin aloittamista täytyi suorittaa Universal Robotsin nettisivuilta löytyvät koulutusmoduulit. Moduuleissa perehdytään robotin liikkumismenetelmiin, turvallisuusmääräyksiin sekä työkalujen hallintaan.

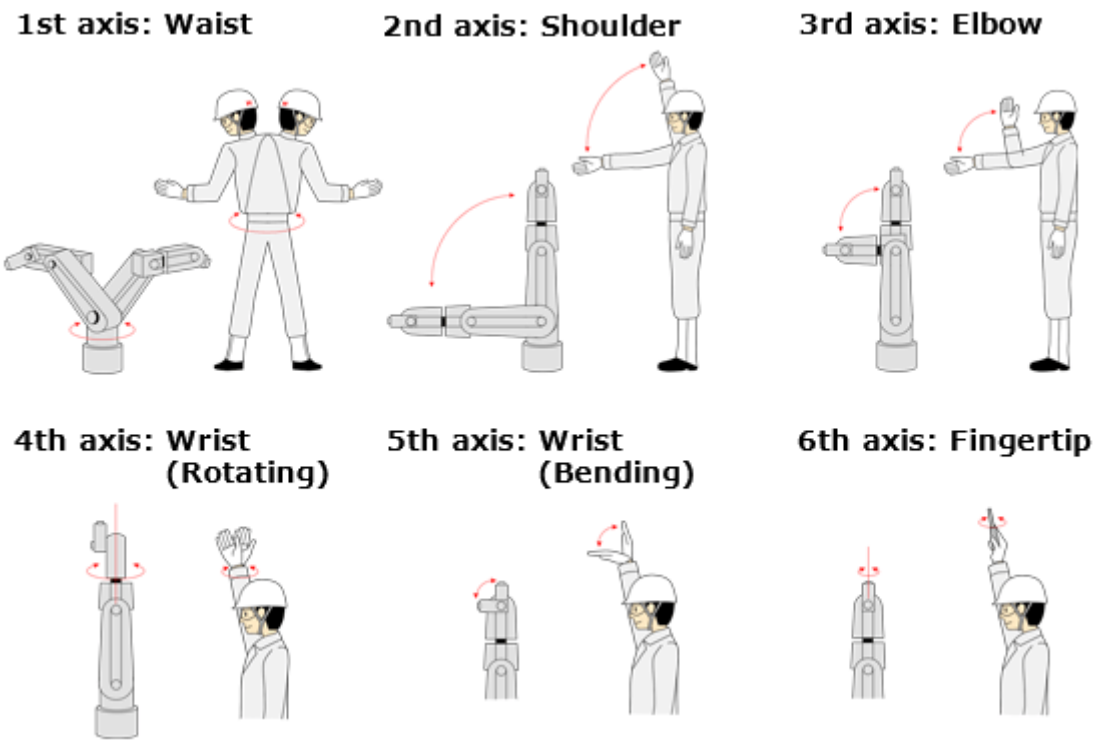
2 TEKNOLOGIAT

2.1 Robotin malli

Projektissa käytetään Oulun ammattikorkeakoulun Linnanmaan kampuksella olevaa Universal Robotsin UR16e-yhteistyörobottia. Kampuksella on myös käytettävissä UR5e-mallinen yhteistyörobotti, joka on pienempi UR16e:hen verrattuna.

Projektin alkuvaiheessa harkittiin oman SO-101-käsivarsirobotin rakentamista hyödyntämällä TheRobotStudion GitHub repositoriota. Sieltä olisi saanut tiedostot robotin rungon 3D-tulostamista varten ja listan tarvittavista moottoreista ynnä muista osista. (TheRobotStudio s.a.) Jotta hinta pysyisi kohtalaisena, osat olisi pitänyt tilata Kiinasta, mutta toimitusajat osille olisivat olleet liian pitkät projektin aikataululle.

UR16e on 6-akselinen käsivarsirobotti. Tämä tarkoittaa, että robotti pystyy kääntymään kuudella eri tavalla, joka mahdollistaa käsivarren liikkumisen ihmiskäden tapaan. Ensimmäiset kolme akselia vastaavat ihmisen vyötäröä ja käsivartta, kun taas viimeiset kolme vastaavat rannetta ja sormenpäitä (KUVA 1).



KUVA 1. 6-akselinen yhteistyörobotti verrattuna ihmiskäteen (Kawasaki Robotics 22.5.2018)

UR16e-robotin hyötykuorma on 16 kg, mikä on paljon enemmän kuin mitä tarvitaan shakkinappuloiden nostamiseen. Robotin ulottuvuus on myös sen verran suuri, että siitä ei aiheudu ongelmia. (TAULUKKO 1.)

TAULUKKO 1. UR16e-robotin tekniset tiedot (Universal Robots s.a. a)

Hyötykuorma	16 kg
Ulottuvuus	900 mm
Vapausasteet	6 pyörivää niveltä
Asennon toistotarkkuus	± 0,05 mm

2.2 Robotiq-lisäosat

Projektissa hyödynnettiin yhteistyörobotissa kiinni olevia Robotiq Hand-E-tarttujaa ja Wrist Camera -lisäosaa. Vaihtoehtona tarttujalle olisi ollut myös Robotiq 2F-85-tarttuja, mutta se oli kooltaan liian suuri. Tarttujan sormet eivät testatessa mahtuneet nostamaan shakkinappulaa laudalta osumatta viereisiin nappuloihin.

Nostaessa nappulaa ylös Hand-E mahtuu nappaamaan sen niin, että molemmille puolelle jää noin 3–4 mm tilaa ennen kuin se osuisi viereisiin nappuloihin. Tästä ei kuitenkaan tarvinnut huolehtia sillä UR16e-robotin asennon toistotarkkuus on ± 0,05 mm (TAULUKKO 1) eli sen pitäisi aina mahtua nappuloiden väliin.

KUVA 2. Robotiq Hand-E-tarttuja (Robotiq s.a. a)

Robotiq Wrist Camera on Universal Robotsille suunniteltu kompakti näköjärjestelmä. Kamerassa on 5 megapikselin värisensori ja elektroninen tarkennus. (Robotiq s.a. b.) Käsivarsirobotti sijoitetaan niin, että tämä lisäosa pystyy liikkeiden välissä kuvaamaan shakkilaudan ylhäältä päin. Kuvasta 3 nähdään, miten lisäosa kiinnittyy UR16e-robottiin.

KUVA 3. Robotiq Wrist Camera (Robotiq s.a. b)

2.3 Robotin kanssa kommunikointi

Yhteistyörobotin ohjaus toteutettiin hyödyntämällä ulkoista kommunikaatiorajapintaa. Kehityksen alkuvaiheessa robottia kokeiltiin ohjata lähettämällä Universal Robotsin omaa URScript-koodia suoraan TCP/IP-yhteyden kautta. Tämä osoittautui kuitenkin haasteelliseksi ja rajoittavaksi, koska lähettäessä komentoja koodin kautta robotti ei odottanut edellisen komennon suorittamista loppuun. Tämän takia robotti suoritti vain viimeiseksi lähetetyn komennon. Tämä ongelma

ratkaistiin lisäämällä jokaisen komennon väliin sleep-komento, joten robotti kerkesi suorittaa liikkeen ennen seuraavan lähettämistä. Tämä tapa ei ollut kuitenkaan optimaalinen tapa robotin hallintaan.

Ratkaisuksi robotin hallintaan löydettiin avoimen lähdekoodin ur_rtde-ohjelmistokirjasto (Lindvig, Iturrate, Kindler & Sloth 2025, 1118–1123). Kirjasto helpottaa kommunikaatiota Universal Robotsin RTDE-rajapinnan (Real-Time Data Exchange) kanssa. RTDE on protokolla, joka mahdollistaa reaaliaikaisen kommunikaation robotin ohjaimen ja ulkokoisten järjestelmien välillä. Protokollan avulla käyttäjä voi sekä lähettää komentoja robotille että vastaanottaa siltä dataa minimaalisella viiveellä, mikä tekee siitä erittäin sopivan tarkkaa ajoitusta vaativiin sovelluksiin. (Universal Robots s.a. b) Tässä työssä ur_rtde-kirjasto auttoi kääntämään Python-kieliset komennot RTDE-rajapinnan ymmärtämään muotoon.

2.4 Robotiq-lisäosien kanssa kommunikointi

Toinen haaste alkuvaiheessa oli robottiin kiinnitetyn Robotiq Hand-E-tarttujan hallinta. Tarttujan hallinta URScriptin kautta ei onnistunut, koska sen avulla ei pystynyt ohjaamaan suoraan Robotiqin tekemiä lisäosia. Tarttujan ohjaus ratkaistiin hyödyntämällä Sam Raspin kehittämää valmista robotiq_gripper-luokkaa (Rasp s.a.). Tämä luokka kommunikoi yhteistyörobotin sisällä olevan Robotiq_gripper UR Cap -ohjelmiston avaaman 63352 portin kautta tarttujan kanssa. Luokan avulla tarttujan hallinta onnistui Python komentojen kautta, mikä ratkaisi haasteet tarttujan operoinnista.

Robotiq Wrist Camera -lisäosan kanssa kommunikoidaan hakemalla sen viimeisin ottama kuva osoitteesta ”http://<robotin_ip_osoite>:4242/current.jpg?type=color”. Osoitteesta saadaan GET-pyynnöllä 640x640 pikselin kokoinen kuva, jota käytetään projektissa shakkiliikkeiden havaitsemiseen. Koska 640x640 ei ole kovin suurilaatuinen kuva verrattuna siihen, että Wrist Camerassa on 5 megapikselin (2560x1920) kamera. Parempilaatuista kuvaa yritettiin saada käyttöön yhdistämällä kamera suoraan tietokoneeseen robotin sijasta. Tämä olisi mahdollistanut täysilaatuisen kuvan käytön ja kameran asetusten muuttamisen, mutta tämä ei kuitenkaan onnistunut.

2.5 Shakkimoottori

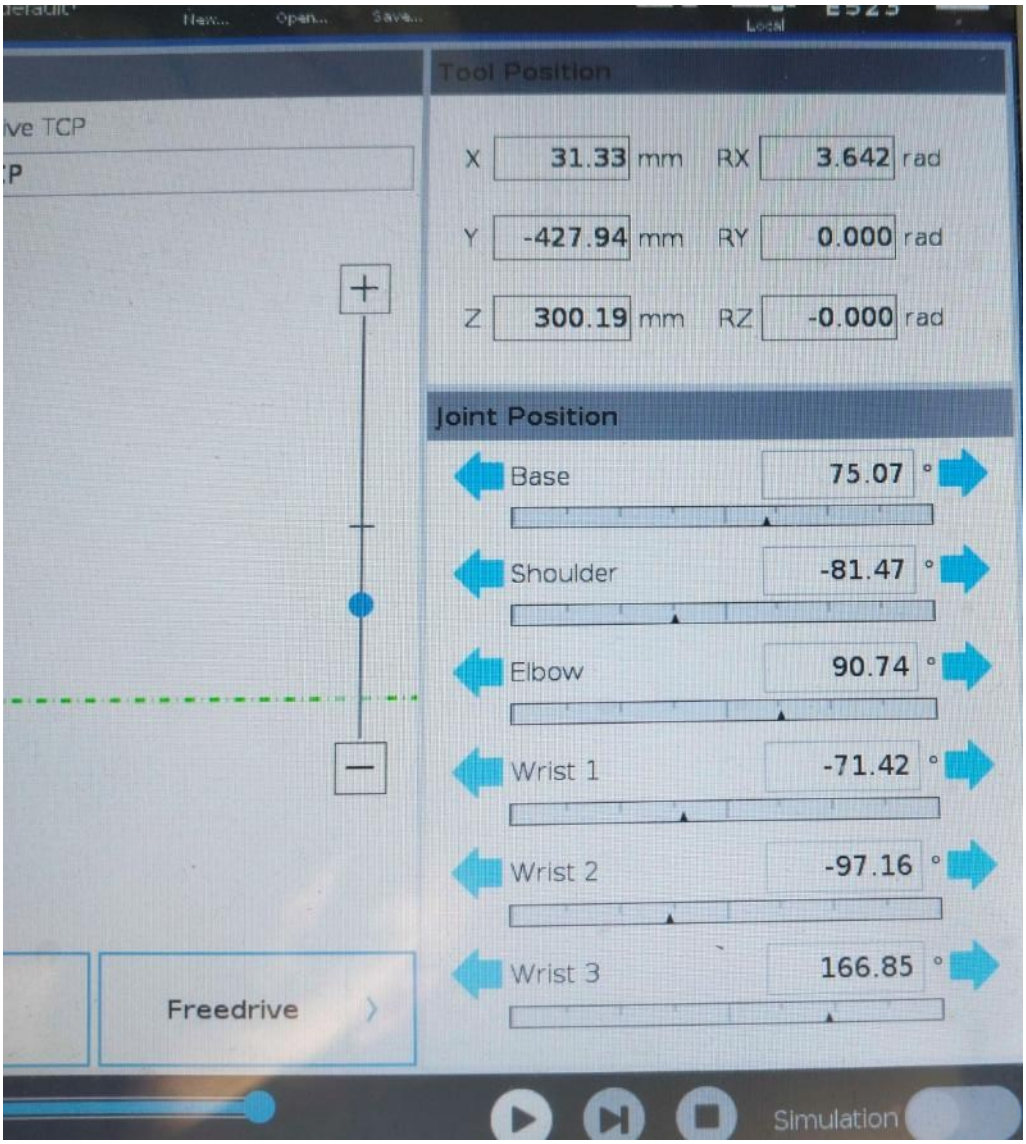
Projektissa käytetään Stockfishiä (Stockfish s.a.), joka on ilmainen avoimen lähdekoodin shakkimoottori. Stockfish on maailman vahvin shakkimoottori ja sen kehitti Tord Romstad, Marco Costalba ja Joona Kiiski. (Chess.com 2020.) Se valittiin tähän projektiin sen helppokäyttöisyyden takia. Koska Stockfish on tunnetuin shakkimoottori, sen käyttöönottoon eri ohjelmointikielillä on tarjolla paljon apuvälineitä ja dokumentaatiota. Projektissa Stockfishin integraation Python-koodiin toteutettiin hyödyntämällä python-chess-kirjastoa.

Kirjasto auttaa kommunikoimaan shakkimoottorin kanssa käyttäen UCI-protokollaa (Universal Chess Interface). Kirjasto sisältää myös ominaisuuden siirtojen validointiin ja muihin shakkilaudan tilan tarkasteluun, kuten shakkauksen, pattitilanteiden ja tasapelien havaitseminen. (Fiekas 2024.) UCI-protokollassa liikkeet ilmoitetaan nappulan lähtö- ja kohderuudulla. Esimerkki tästä on e2e4, jossa nappula liikkuu ruudusta e2 ruutuun e4. Nappuloiden lyöntejä ei merkitä erikseen vaan niiden havaitsemisen hoitaa shakkimoottori ja kirjasto. Korotukset merkitään lisäämällä liikkeen perään joko q, r, b tai n ilmoittamaan mihin nappulaan sotilas korotettiin. Esimerkki korotuksesta on e7e8q, jossa sotilas liikkuu ruudusta e7 ruutuun e8 ja korotetaan kuningattareksi. (McCabe s.a.).

3 ENSIMMÄISEN VERSION TOTEUTUS

3.1 Nappuloiden liikuttaminen

Ennen ensimmäisen konenäköversion kehittämistä tavoitteena oli yksinkertaisesti shakkinappuloiden liikuttaminen tarkasti ympäri shakkilautaa ruudusta toiseen. Tätä varten täytyi määrittää laudan jokaiselle ruudulle omat koordinaatit, joka tehtiin asettamalla robotti niin, että tarttujan kärki on shakkilaudan a1-ruudun keskellä. Kun robotti on oikeassa asennossa, ruudun X-, Y- ja Z-koordinaatit voidaan lukea sen käyttöliittymästä (KUVA 4) ja muiden ruutujen koordinaatit voidaan laskea niiden avulla. Nämä koordinaatit tallennettiin Python sanakirjaan, josta ne noudetaan aina tarvittaessa.



KUVA 4. Esimerkki tarttujan koordinaateista

Kun kaikki koordinaatit ovat tiedossa, shakkinappuloita pystyy liikuttamaan Python-koodin avulla. (KOODIESIMERKKI 1) Koodille annetaan kahden ruudun koordinaatit, joita hyödyntämällä se ohjaa robotin liikuttamaan nappulan ruudusta toiseen.

```
Koodi alkaa.

# Move above starting square
rtde_c.moveL([start_x, start_y, start_z + 0.1, start_rx, start_ry, start_rz],
0.20, 1)

# Move down to pick up the piece
rtde_c.moveL([start_x, start_y, start_z, start_rx, start_ry, start_rz], 0.20,
0.4)

# Close the gripper
```

```
gripper.move_and_wait_for_pos(255, 100, 50)

# Move back up with the piece
rtde_c.moveL([start_x, start_y, start_z + 0.1, start_rx, start_ry, start_rz],
0.20, 1)

# Move above the destination square
rtde_c.moveL([end_x, end_y, end_z + 0.1, end_rx, end_ry, end_rz], 0.20, 1)

# Move down to place the piece
rtde_c.moveL([end_x, end_y, end_z, end_rx, end_ry, end_rz], 0.20, 0.4)

# Open the gripper
gripper.move_and_wait_for_pos(155, 255, 50)

# Move back up above the destination square
rtde_c.moveL([end_x, end_y, end_z + 0.1, end_rx, end_ry, end_rz], 0.20, 1)

# Move robot to home position
rtde_c.moveL(get_square_coords("home"), 0.20, 1)

Koodi päättyy.
```

KOODIESIMERKKI 1. Shakkinappulan ruudusta ruutuun siirtämiseen vaadittavat liikekomennot

3.2 Shakkilogiikka

Seuraavana vaiheena on shakkilogiikan implementointi. Tässä hyödynnetään python-chess-kirjastoa, jonka avulla pistettiin aluksi Stockfish-shakkimoottori pelaamaan itseänsä vastaan ja liikuttamaan nappuloita robotilla. Kirjasto auttaa kommunikoimaan Stockfishin kanssa saadakseen siltä liikkeen, joka annetaan robotille liikekomentona (KOODIESIMERKKI 2).

```
Koodi alkaa.

# Initialize board
board = chess.Board()

#Loop until the game is over
while not board.is_game_over():

    # Get move from Stockfish
    result = engine.play(board, chess.engine.Limit(time=1.0)
    stockfish_move = result.move

    # Parse the move and call movePiece
    movePiece(stockfish_move.uci()[:2], stockfish_move.uci()[2:])

    # Update the board state with the move
    board.push(stockfish_move)
```

```
Koodi päättyy.
```

KOODIESIMERKKI 2. Stockfish pelaa itseään vastaan ja liikuttaa yhteistyörobottia

Koodi näyttää alussa onnistuvan nappuloiden liikuttelussa hyvin, mutta ongelma ilmestyy Stockfishin tehdessä linnoitusliikkeen. Linnoitus ilmoitetaan UCI-protokollassa kuninkaan liikkeenä, esimerkiksi ”e1g1” tai ”e8c1”. Koodin movePiece-funktio ei tässä tapauksessa liikuttanut tornia shakkilaudalla, minkä vuoksi myöhemmin pelissä robotti tarttuu ilmaan yrittäessään nostaa tornia tai yrittää pistää toisen nappulan tornin päälle.

Ratkaisu tähän löytyy python-chess-kirjaston is_castling-funktiosta, jonka avulla voidaan tarkistaa, onko annettu liike linnoitus vai ei. Kun is_castling on tosi, täytyy tarkistaa mikä neljästä mahdollisesta linnoitusliikkestä (e1g1, e1c1, e8g8, e8c8) on kyseessä. Kun joku noista liikkeistä havaitaan koodissa, kutsutaan kuninkaan liikkeen lisäksi movePiece-funktiota linnoitusta vastaavalle tornille. Muiden nappuloiden lyönti ja ohestalyönti hoidetaan samankaltaisesti is_capture- ja is_en_passant-funktion avulla. Lyöty nappula siirretään ensin pois laudalta ennen uuden nappulan tilalle siirtämistä.

Korottamisen tarkistamiselle ei ole omaa funktiota, mutta ne tilanteet hoidetaan tarkistamalla, onko UCI-liikkeen perässä mikään korotusta tarkoittava kirjan (q, r, b, n). Jos korotus havaitaan, robotti ohjataan nostamaan laudan vierellä oleva ylimääräinen nappula. Käytössä on vain yksi ylimääräinen kuningatarnappula, joten muihin nappuloihin korottaessa käytetään myös samaa nappulaa. Muihin nappuloihin ei kuitenkaan ole syytä korottaa useimmissa tapauksissa. Jos pelin aikana tapahtuu useampia korotuksia, täytyy ottaa yksi pelin aikana lyödyistä nappuloista ja asettaa se paikalle, josta robotti voi nostaa sen käyttöön.

3.3 Ensimmäinen konenäkö

Projektin seuraava vaihe on liikkeiden havaitseminen Robotiq Wrist Camera -lisäosan ottamien kuvien avulla. Ideana on ottaa ylhäältäpäin kuva shakkilaudan tilanteesta ennen ja jälkeen pelaajan tekemää liikettä. Kuvia verrataan toisiinsa

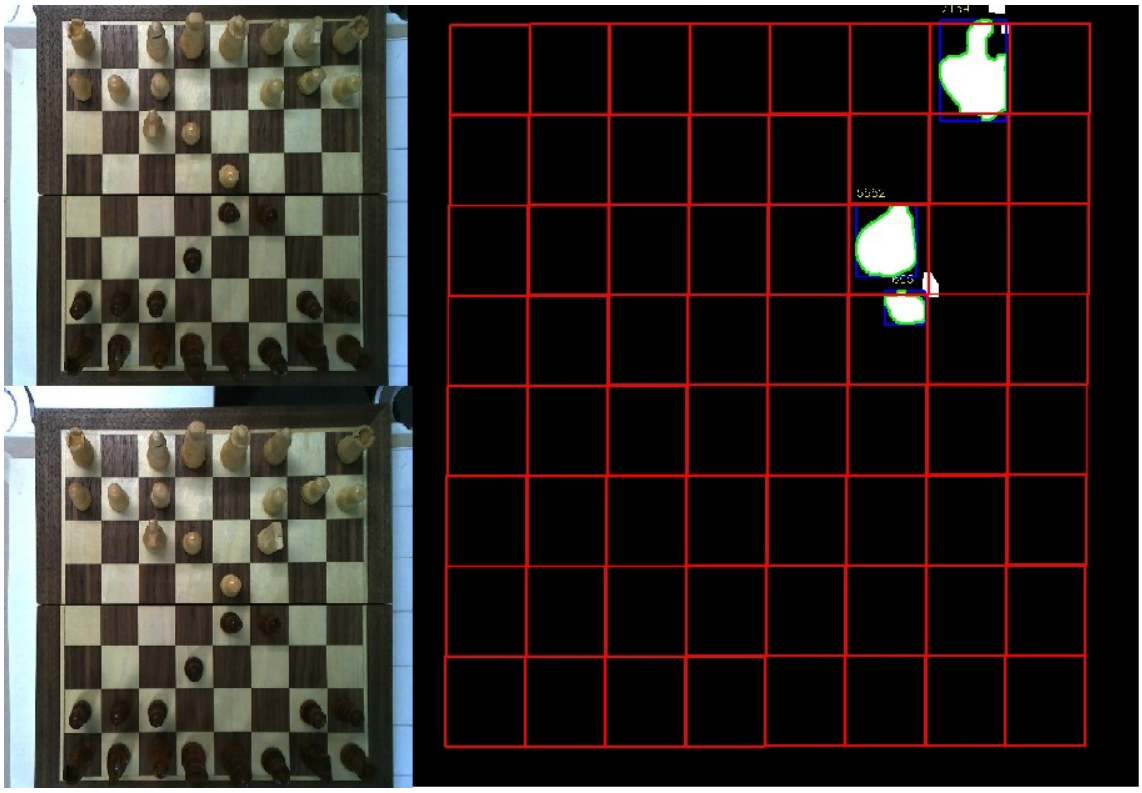
ja niiden välisiä eroja käytetään tunnistamaan tehty shakkiliike. Vertailuun käytetään OpenCV-kirjaston absdiff-funktiota, joka näyttää kahden kuvan välisen eron.

```
Koodi alkaa.

# Convert image to grayscale
img1 = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
img2 = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Blur the image
img1 = cv2.GaussianBlur(img, (5,5), 0)
img2 = cv2.GaussianBlur(img, (5,5), 0)

# Calculate difference
diff = cv2.absdiff(img1, img2)

# Apply thresholding
_, threshold = cv2.threshold(diff, 60, 255, cv2.THRESH_BINARY)

# Find contours
contours, _ = cv2.findContours(threshold, cv2.RETR_EXTERNAL, cv2.CHAIN_AP-
PROX_SIMPLE)

Koodi päättyy.
```

KOODIESIMERKKI 3. Kuvien prosessointi ja erojen vertailu

Kuvat muutetaan ensin harmaansävyisiksi ja sumennetaan käyttämällä cvtColor- ja GaussianBlur-funktiota, mikä auttaa vähentämään kuvien kohinaa. Tämän jälkeen kuvat syötetään absdiff-funktiolle saadaksemme niiden välisen eron. Saatuun kuvaan käytetään vielä threshold-funktiota, joka muuttaa kaikki tietyn harmaasävyarvon yläpuolella olevat pikselit joko valkoisiksi tai mustiksi. Lopuksi hyödynnetään findContours-funktiota, joka tunnistaa kuvassa olevien kuvioiden ääriviivat. (KOODIESIMERKKI 3.)

Kun ääriviivat saatiin havaittua, kuva jaettiin 8x8 ruudukkoon ja katsottiin missä ruudussa kuvioita havaittiin (KUVA 5). Kuvasta huomataan miten varjot voivat aiheuttaa häiriöitä siirron havaitsemisessa. Varjon ääriviivat ulottuvat nappulan viereiselle ruudulle, mikä aiheuttaa liikkeen virhelukeman riippuen varjon koosta.



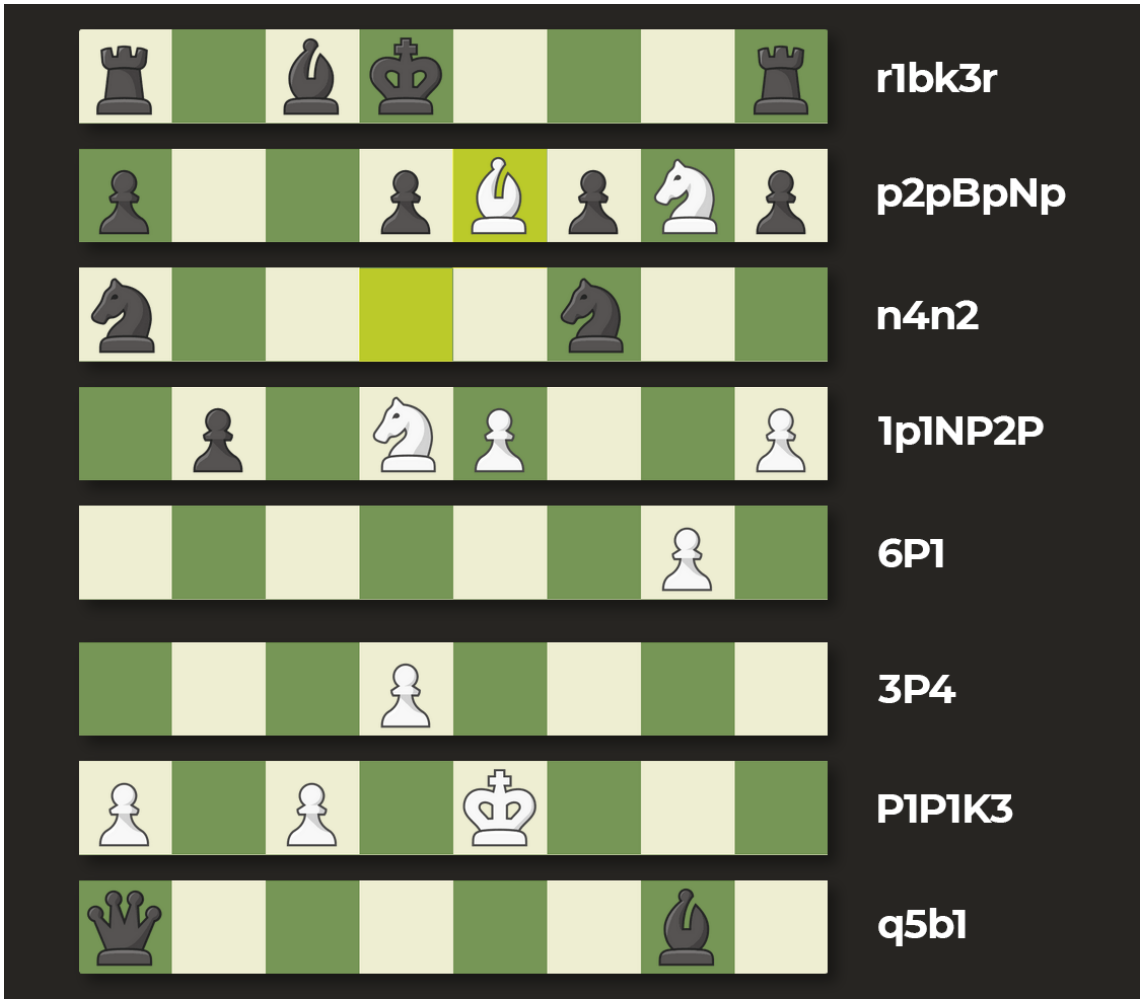
KUVA 5. Koodi havaitsee ratsun liikkeen laudalla

Virhelukemien poistamiseen kokeiltiin useita keinoja, kuten erilaisen sumentamisfunktion käyttö, kuvan kontrastin vaihtaminen, threshold-funktion arvojen muutto ja dilate-funktion käyttö laajentaakseen kuvassa olevia alueita yhtenäisiksi esineiksi. Nämä muutokset vähentävät virhelukemia, mutta eivät kuitenkaan poistanut niitä kokonaan.

Koska virheitä sattui sen verran useasti, haluttiin ominaisuus tallentaa shakkipeli muistiin, jotta sitä voisi jatkaa siitä tilanteesta mihin se oli jäänyt ennen ohjelman sulkemista. Tämän avulla koodiin pystyi tekemään pieniä muutoksia ilman, että shakkinappulat pitäisi siirtää aloituspaikoilleen joka kerta. Peli tallennettiin ottamalla muistiin shakkilaudan tilanne FEN-merkkijonona (Forsyth-Edwards Notation), joka kuvaa pelitilanteen yhdessä tekstirivissä.

FEN-merkkijonon ensimmäinen kenttä määrittelee nappuloiden sijainnin shakkilaudalla. Sijainnit kuvataan riveittäin a-linjalta h-linjalle alkaen kahdeksannesta rivistä ja päättyen ensimmäiseen riviin. Mustat nappulat merkitään pienillä kirjaimilla ja valkoiset nappulat suurilla kirjaimilla. Nappulat merkataan kirjaimilla p, r, n, b, q ja k (sotilas, torni, ratsu, lähetti, kuningatar, kuningas). Tyhjät ruudut

ilmaistaan numeroilla yhdestä kahdeksaan riippuen siitä, kuinka monta tyhjää ruutua nappuloiden välissä on peräkkäin. Esimerkki FEN-merkkijonon nappuloiden sijaintien merkinnästä on ”r1bk3r/p2pBpNp/n4n2/1p1NP2P/6P1/3P4/P1P1K3/q5b1” (KUVA 6). FEN-merkkijonon merkataan myös sijaintien lisäksi aktiivinen väri, linnoitusoikeudet, ohestalyöntiruutu, puolisiirtolaskuri ja siirtonumero. Nämä ilmoitetaan lisäämällä merkkijonon loppuun ”b KQkq e3 0 1”, joka tarkoittaa tässä tapauksessa, että on mustan pelaajan vuoro, molemmilla on kummatkin linnoitusoikeudet ja ohestalyönti on mahdollista ruudussa e3. (Chess.com 2020b.)



KUVA 6. Esimerkki FEN-merkkijonon nappuloiden sijainnin merkinnästä (Chess.com 2020b)

Toisen opiskelijan ehdotuksesta shakkilaudan päälle pistetiin valkoinen paperinpala (KUVA 7). Tämä auttaa parantamaan nappuloiden ja laudan välistä kontrastia ja vähentää valon aiheuttamaan kiiltoa laudan päällä. Vähentynyt kiilto

näyttäisi myös muuttavan kameran automaattivalotusasetuksia johtaen himmeämpään kuvaan. Paperi paransi liikkeiden havaitsemista, mutta varjot saattoivat vieläkin aiheuttaa virhelukemia. Virhelukemista huolimatta robottia vastaan on mahdollista pelata shakkipeli alusta loppuun.

KUVA 7. Shakkilaudan päällä valkoinen paperi

4 KOULUTETTUA KONENÄKÖMALLIA KÄYTTÄVÄ TOTEUTUS

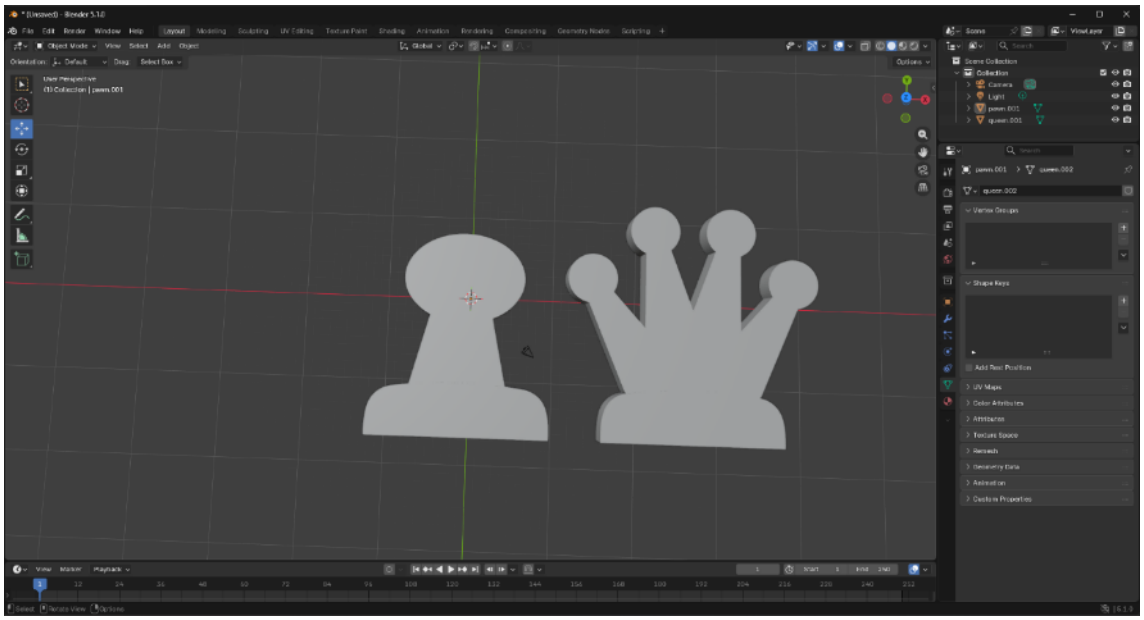
Aikaisemman version havaittiin olevan liian riippuvainen valaistuksesta ja hyvässä valaistuksessakin virhelukemia saattoi esiintyä useasti pelin aikana. Tästä syystä päätettiin kouluttaa oma konenäkömalli YOLO (You Only Look Once) -koneoppimisalgoritmiä hyödyntäen. Uskon, että perinteisten shakkinappuloiden käyttäminen kouluttamisessa olisi tuottanut vaikeuksia ja aikaa olisi kulunut enemmän. Perinteisiä nappuloita olisi pitänyt kuvata sivulta, mikä johtaisi nappuloiden osittaiseen peittymiseen toisen nappulan taakse, minkä tarkka havaitseminen olisi vaatinut suuren määrän kuvia koulutusta varten.

4.1 Uudet nappulat

Helpottaakseni koulutusta, päätin suunnitella ja 3D-tulostaa omat shakkinappulat. Nappuloiden tekemiseen käytettiin pohjana käyttäjän Drawliphant Printables sivustolle julkaisemia 3D-malleja (KUVA 8). Nappuloiden päälle oli suunniteltu nappulaa vastaa symboli, mikä oli kouluttamista miettien hyödyllinen ominaisuus. Käyttäjän julkaisemissa malleissa sotilasnappula muistutti kuitenkin perinteistä muotoaan, eli ylhäältä kuvattuna ei ollut symbolia. Muiden nappuloiden varret olivat myös turhan yksityiskohtaisia, joten päätettiin luoda omat nappulat näitä malleja pohjana käyttäen.

KUVA 8. Käyttäjän Drawliphant suunnittelemat shakkinappulat (Drawliphant 26.8.2025)

Mallien STL-tiedostot ladattiin verkkosivulta ja avattiin Blender 3D-mallinnusohjelmalla. Siellä nappuloista leikattiin irti vain päällä olevat symbolit erikseen tulostettavaksi. Koska tiedostot eivät sisältäneet sotilaan symbolilla varustettua nappulaa, se mallinnettiin itse leikkaamalla kuningatarsymbolin kruunusta osa irti. Leikkaamisen jälkeen osa suurennettiin ja liitettiin sen omaan alustaan (KUVA 9). Tämän jälkeen kaikki symbolit ja sylinterin muotoiset alustat tulostettiin 3D-tulostimella. Tulostuksen jälkeen symbolit liimattiin omiin alustoihin. Uusien nappuloiden valmistuttua tehtiin myös isompi shakkilauta piirtämällä ruudut valkoiselle paperille. Isompi lauta auttaa vähentämään tarttujan vahingollisia törmäyksiä nappuloihin.



KUVA 9. Irrotettu kuningatarsymboli ja sen osasta tehty sotilassymboli Blender-ohjelmassa (Juuso Korpinen, 2026)

4.2 Konenäkömallin koulutus

Tarvitsin kouluttamista varten suuren määrän kuvia laudasta eri tilanteissa, joten pistin Stockfishin pelaamaan robotilla itseään vastaan. Näin minun ei itse tarvinnut liikutella nappuloita eri tilanteisiin, mikä säästi huomattavasti aikaa. Jokaisen liikkeen jälkeen Wrist Camera otti kuvan ja tallensi sen omaan kansioon. Asetin robotin aluksi pelaamaan kaksi peliä eri valaistuksessa. Sain ensimmäisestä pelistä 153 kuvaa, mutta toinen peli loppui nopeasti ja sain vain 60 kuvaa. Kokeilua varten nämä kuvat ovat kuitenkin riittävät.

Jotta kuvia voidaan käyttää konenäön koulutukseen, ne piti ensin annotoida. Annotointi tarkoittaa kuvien merkkaamista metadatalla, jonka avulla koneoppimisalgoritmi oppii tunnistamaan halutut esineet. Ennen annotoinnin aloittamista täytyi ensin määritellä tarvittavat labelit, eli luokat. Shakissa on 12 eri nappulatyyppiä, joten jokaiselle määriteltiin vastaava luokan nimi (TAULUKKO 2). Itse annotointi toteutettiin käyttämällä rajauslaatikoita, jotka ovat esineen ympärille piirrettyjä suorakulmaisia laatikoita.

TAULUKKO 2. Luokat ja niitä vastaavat nappulat

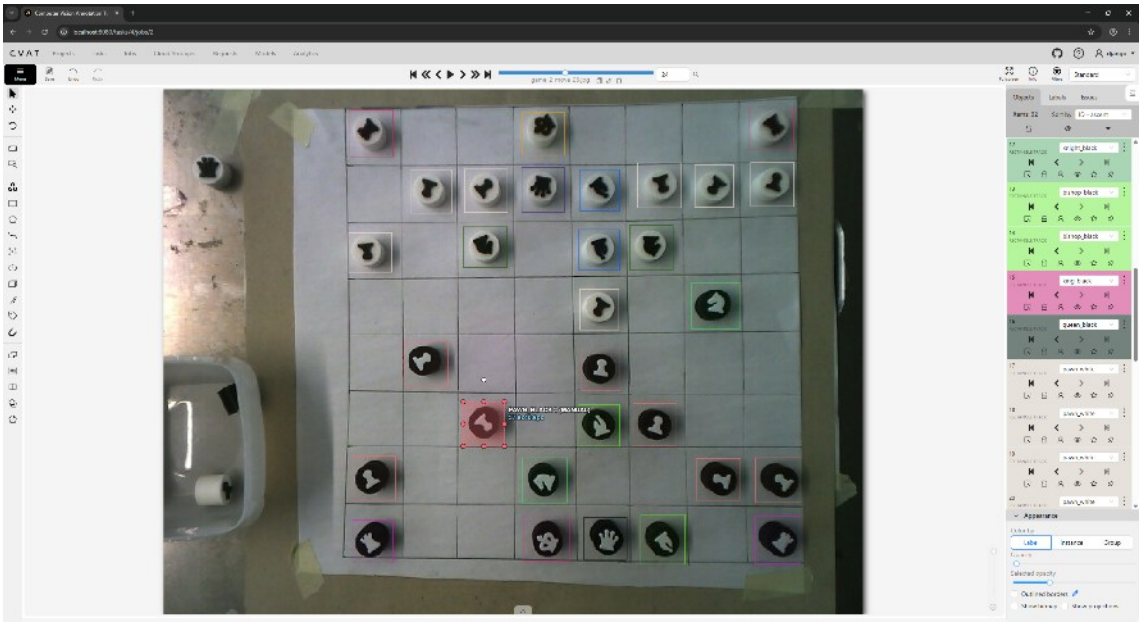
Luokan nimi	Vastaava nappula
pawn_black / pawn_white	Musta / Valkoinen sotilas
knight_black / knight_white	Musta / Valkoinen ratsu
rook_black / rook_white	Musta / Valkoinen torni
bishop_black / bishop_white	Musta / Valkoinen lähetti
queen_black / queen_white	Musta / Valkoinen kuningatar
king_black / king_white	Musta / Valkoinen kuningas

Aloitin annoitoimisen käyttämällä LabelImg-annotointityökalua, mutta sen toiminnallisuus ei soveltunutkaan hyvin shakkipelien merkkaamiseen. Kun olin annotoinut ensimmäisen kuvan ja siirryin seuraavaan, minun olisi pitänyt piirtää kaikki 32 rajauslaatikkoa uudestaan. Tarvitsin siis annoitointityökalun, joka mahdollistaa edellisen kuvan merkkausten siirtämisen seuraavaan kuvaan. Tutkittuani eri vaihtoehtoja päädyin valitsemaan CVAT-annotointityökalun, jota käytetään usein videoiden annotoimiseen. Sillä on siis haluamani ominaisuus kopioida merkinnät kuvasta toiseen.

CVAT tarjoaa kolme eri versiota työkalusta, joista yksi on maksullinen ja yrityksille suunnattu CVAT Enterprise, toinen on pienemmille tiimeille tarkoitettu pilvipalvelu CVAT Online ja viimeinen on itse isännöity ja täysin ilmainen CVAT Community. Online versiota on mahdollista käyttää ilmaiseksi, mutta joitain ominaisuuksia ja tallennustilaa rajoitetaan. En tiennyt paljonko tallennustilaa tulen tarvitsemaan, joten valitsin Community version työkalusta. Tätä versiota käyttääkseni täytyi ladata Docker Deskop, jonka avulla pystyin itseisännöimään työkalun.

Annotoiminen aloitettiin taas piirtämällä ensimmäiseen kuvaan jokaisen nappulan ympärille luokalla merkattu rajauslaatikot. CVAT Communityn ominaisuuksien takia seuraavaan kuvaan siirryttäessä edellisen kuvan rajauslaatikot

automaattisesti kopioitiin uuden kuvaan. Tämän jokaisessa seuraavassa kuvassa täytyi siirtää vain yhtä rajauslaatikkoa uuteen paikkaan. Poikkeuksena tähän olivat kuvat, joissa oli tapahtunut nappulan lyönti. Näissä kuvissa täytyi liikuttaa nappulan rajauslaatikko uuteen ruutuun ja samalla poistaa lyödyn nappulan laatikko.



KUVA 10. CVAT Communitylla tehty annotaatio shakkipelin tilanteesta (Juuso Korpinen, 2026)

Kun koko kuvamateriaali saatiin annotoitua se jaettiin satunnaisesti opetus- ja validointiaineistoon. Opetusaineisto viittaa kuviin, joita käytetään mallin kouluttamiseen ja suhteiden oppimiseen, mikä mahdollistaa ennusteiden tekemisen myöhemmin. Validointiaineisto puolestaan toimii itsenäisenä aineistona mallin suorituskyvyn vertailuun ja ylisovittumisen ehkäisemiseen mikä auttaa parhaan mallin valinnassa. Ylisovittuminen tarkoittaa tilannetta, jossa malli on oppinut opetusaineiston yksityiskohdat niin tarkasti, ettei se kykene enää yleistämään oppimaansa aineiston ulkopuolisiin kuviin. (Wilber & Werness s.a.)

Itse kouluttaminen on mallin tekemisessä helpoin osa. Ultralytics-kirjastolla se vaati vain muutaman rivin Python-koodia (KOODIESIMERKKI 4). Koodissa tuodaan aluksi käyttöön Ultralytics-kirjasto sekä esikoulutettu YOLO26-malli. Sen jälkeen aloitetaan mallin kouluttaminen valmiilla koulutusfunktiolla, jolle annetaan kolme parametria. Data-parametri määrittää polun YAML-tiedostoon, joka kertoo

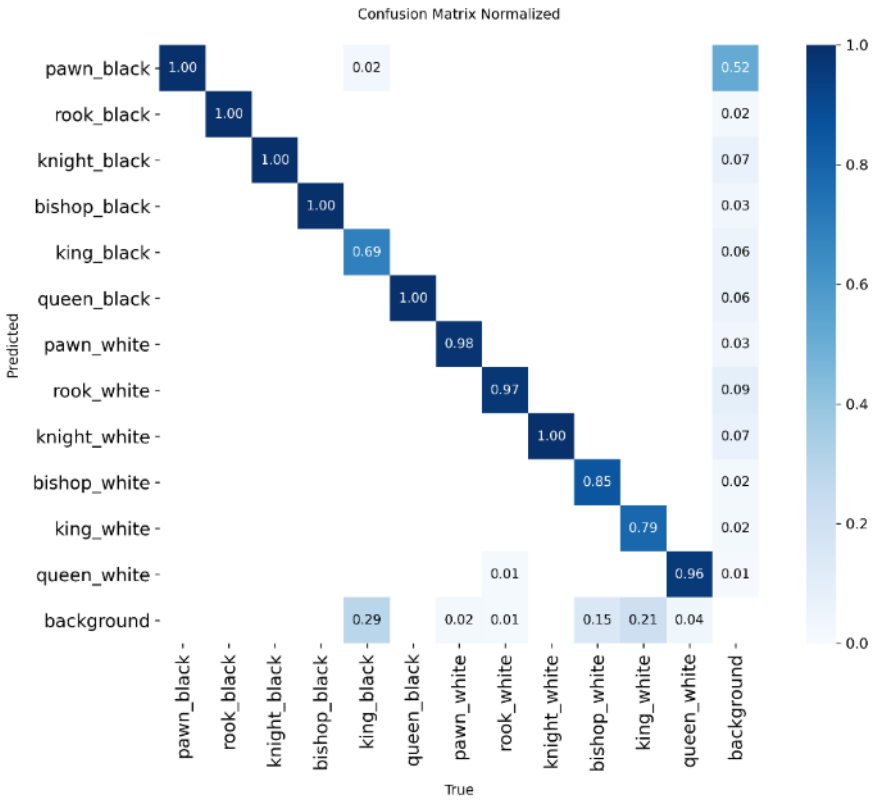
mallille opetus- ja validointiaineistojen sijainnit sekä luokkien nimet. Epochs-parametri määrittää epookkien eli koulutuskierrosten määrän, joka asetettiin sataan. Malli käy siis opetusaineiston läpi 100 kertaa. Viimeiseksi annettiin imgsz-parametri, joka määrittää kuvakooksi 640 pikseliä. Kaikki syötettävät kuvat skaalataan tuohon kokoon ennen käyttöä. Koulutuksen päätyttyä kirjasto tallentaa automaattisesti parhaat painoarvot sisältävän tiedoston sekä viimeisimmän kierroksen painoarvot.

```
Koodi alkaa.

from ultralytics import YOLO

# Load a model
model = YOLO("yolo26n.pt") # load a pretrained model (recommended for training)

# Train the model
results = model.train(data="data.yaml", epochs=100, imgsz=640)

Koodi päättyy.
```

KOODIESIMERKKI 4. Konemallin kouluttamiseen käytetty Python-koodi (Ultralytics 2026a)

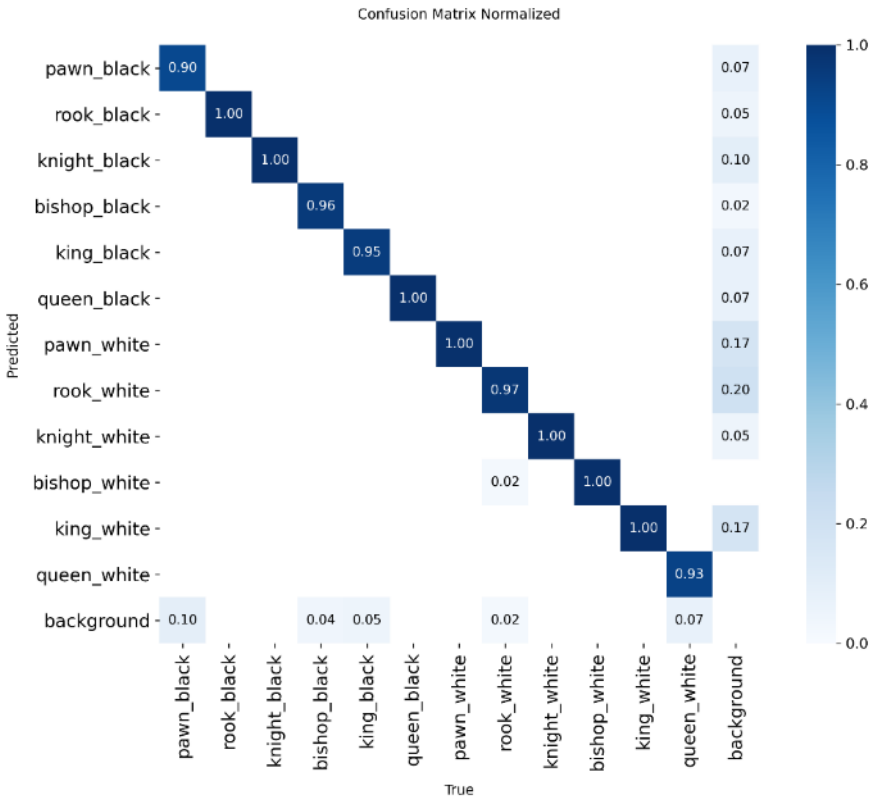
Koulutuksessa käytetty esikoulutettu YOLO26 on 14.1.2026 julkaistu YOLO-sarjan uusin malli, joka on suunniteltu matalatehoisille laitteille. Malli eroaa aikaisemmista versioista poistamalla yhden jälkikäsittelyvaiheen ja hyödyntää uutta optimointialgoritmiä, mikä vähentää viivettä ja lisää vakautta. Nämä ja muut uudistukset mahdollistavat tarkan pienten kohteiden tunnistuksen ja jopa 43 % nopeamman toiminnan prosessoreilla. (Ultralytics 2026b.) Prosessorisuorituskyvyn optimointi on työn kannalta hyödyllinen, koska YOLOn käyttämä näytönohjainkiihdytys vaatii NVIDIA-näytönohjaimen, eikä käytetyn tietokoneen AMD-näytönohjain ei pysty hyödyntämään sitä ominaisuutta.

Kun mallin koulutus on valmis, sen tuloksista muodostettiin automaattisesti eri kaavioita. Luodusta sekaannusmatriisikaaviosta nähdään, että monet nappulat saivat melkein täydellisen 1,00 tuloksen, mutta mallilla on suuria ongelmia tiettyjen nappuloiden kanssa. Mustista sotilaista peräti 52 % sekoittui taustaan ja mustista kuninkaista taustaan sekoittui 29 %. Myös 21 % valkoisista kuninkaista hukkui taustaan. (KUVA 11.) Nämä virhehavainnot johtuivat todennäköisesti huonosti tehdystä annotaatiosta. Annotoiduissa kuvissa rajauslaatikot piirrettiin liian leveästi, joten laatikoiden sisällä on nappulan lisäksi paljon tyhjää tilaa.



KUVA 11. Ensimmäisen mallin sekaannusmatriisi

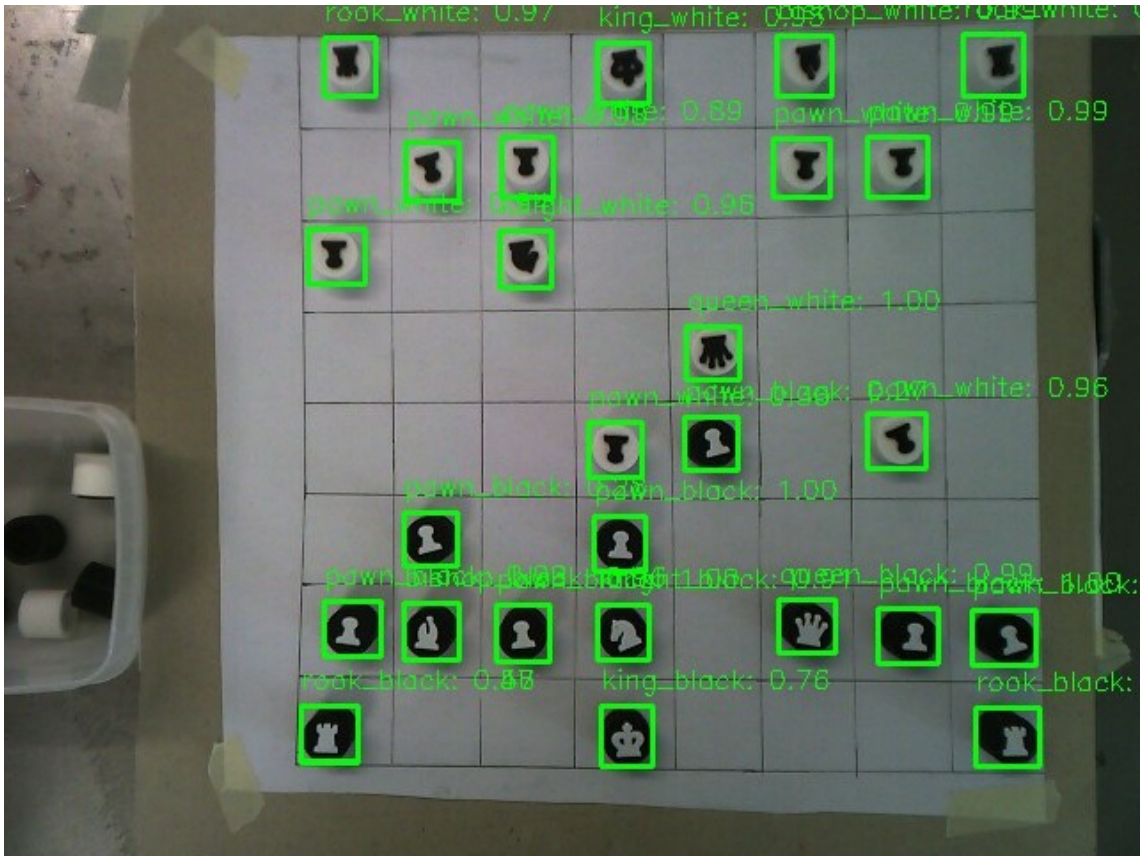
Ongelmien korjaamiseksi kaikki kuvat annotoitiin uudestaan mahdollisimman tiukoilla rajauslaatikoilla. Uudet annotoidut kuvat jaettiin opetus- ja validointiaineistoihin ja niillä koulutettiin uusi malli. Uuden mallin sekaannusmatriisista (KUVA 12) nähdään, että kaikkien nappuloiden tunnistaminen on parantunut. Kaikkien nappuloiden tunnistaminen parantui, eikä niitä sekoiteta taustan kanssa lähes yhtä useasti. Mustien sotilaiden tarkkuus saatiin nostettua 0,90 tasolle ja kuninkaiden virheet minimoitua.



KUVA 12. Toisen mallin sekaannusmatriisi

4.3 Konenäkömallin integrointi

Mallin integroiminen aikaisempaan koodin vaati siihen muutoksia. Kun kuva shakkilaudasta syötetään koodissa konenäkömallille, se palauttaa piirrettyjen rajauslaatikoiden koordinaatit (x1, y1, x2, y2), havaitun nappulan luokan ja varmuusarvon. Koordinaattien avulla voimme piirtää havaituille nappuloille rajauslaatikot ja tarkistaa onko malli tunnistanut kaiken oikein (KUVA 13).



KUVA 13. Konenäkömalli havaitsee laudalla olevat nappulat

Molempien kuvien havaitut nappulat tallennetaan sanakirjoihin vertausta varten (KOODIESIMERKKI 5). Sanakirjoissa olevia nappuloiden koordinaatteja verrataan toisiinsa selvittääkseen mikä niistä liikkui toiseen ruutuun. Kun nappula on tiedossa, voidaan helposti kertoa liike UCI-muodossa shakkimoottorille.

```
Koodi alkaa.

detections = []
    # Iterate over results
    for result in results:
        if result.boxes is not None:
            # Iterate over each box and save data
            for box in result.boxes.data:
                x1, y1, x2, y2, conf, cls = box.tolist()
                detections.append({
                    "bbox": (int(x1), int(y1), int(x2), int(y2)),
                    "class": int(cls),
                    "confidence": float(conf)
                })
Koodi päättyy.
```

KOODIESIMERKKI 5. Konenäkömallin palauttaman datan irrottaminen

5 POHDINTA

Projekti olisi pitänyt aloittaa 3D tulostetuilla nappuloilla ja kouluttaa oma malli heti alussa. Ensimmäinen versio on liian riippuvainen valaistuksesta. Täydellisessä valaistuksessa se todennäköisesti toimisi virheittä, mutta huoneessa, jossa käytetty robotti sijaitsi tämä ei ollut mahdollista. Ensimmäiseen version hienosäätöön tuli käytettyä liikaa aikaa, minkä olisi voinut käyttää toisen version kehitykseen.

Työn lopputulos täyttää johdannossa asetetut tavoitteet. Robottia vastaan on mahdollista pelata shakkipeli alusta loppuun minimaalisilla virheillä hyödyntämällä itse koulutettua konenäkömallia. Uudella laudalla robotin ei myöskään ole mahdollista törmätä nappuloihin, ellei ihminen siirrä nappulaa todella huolettomasti. Jos projektin kehittämiseen olisi ollut enemmän aikaa, olisi haluttu lisätä ominaisuus pelata eri shakkivariantteja, kuten Giveaway- tai Atomic-shakkia. Toinen haluttu ominaisuus olisi ollut mahdollisuus syöttää FEN-merkkijono koodiin, jonka perusteella robotti olisi itse siirtänyt kaikki nappulat oikeille paikoille. Tämä olisi ollut hyödyllinen, jos esimerkiksi haluasi ratkaista shakkipulmia tai kokeilla pelata kuuluisien shakkipelien loppuvaiheita.

Opinnäytetyön aikana opittiin miten yhteistyörobotteja voi soveltaa eri tarkoituksiin. Alkututkinnan aikana huomattiin, miten käsivarsirobotteja käytetään teollisuuden eri alueilla. Niiden käytöstä oppimisesta saattaa olla hyötyä tulevaisuudessakin. Ennen projektia ei ollut käytetty konenäköä, joten oman mallin kouluttamisesta ja käytöstä oli kiinnostavaa oppia.

Lopputulokseen oltiin lisäominaisuuksien puuttumisesta huolimatta tyytyväisiä. Projektia on mahdollista jatkokehittää ominaisuuksilla ja tekemällä siitä helpommin käyttöön otettavan. GitHubiin voisi myös tehdä selkeän dokumentaation, jonka avulla kuka tahansa kampuksella voisi kokeilla projektia helposti.

LÄHTEET

Bradski, G. 2000. The OpenCV Library. Dr. Dobb's Journal of Software Tools, 25, s. 120–125.

Chess.com 2020a. Stockfish - Chess Engines - Chess.com. URL: https://www.chess.com/terms/stockfish-chess-engine. Luettu: 24.5.2026.

Chess.com 2020b. FEN (Forsyth-Edwards Notation) - Chess Terms. URL: https://www.chess.com/terms/fen-chess. Luettu: 24.5.2026.

Drawliphant 2025. Icon Online Chess pieces by Drawliphant | Download free STL model | Printables.com. URL: https://www.printables.com/model/1395595-icon-online-chess-pieces. Luettu: 25.4.2026.

Fiekas, N. 2024. python-chess: a chess library for Python. URL: https://python-chess.readthedocs.io/en/latest/index.html. Luettu: 18.5.2026.

Kawasaki Robotics 2018. How Are Industrial Robots Built? A Guide on the Components and the Movement of Robot Arms. URL: https://kawasakirobotics.com/asia-oceania/blog/1804-03/. Luettu: 15.4.2026.

Lindvig, A.P., Iturrate, I., Kindler, U. & Sloth, C. 2025. ur_rtde: An Interface for Controlling Universal Robots (UR) using the Real-Time Data Exchange (RTDE). Institute of Electrical and Electronics Engineers Inc. Munich, Germany. URL: https://doi.org/10.1109/SII59315.2025.10871000. Luettu: 13.5.2026.

McCabe, L. s.a. Intro - UCI Docs - Obsidian Publish. URL: https://publish.obsidian.md/modern-uci-doc/UCI+Docs/Intro. Luettu: 24.5.2026.

Rasp, S. s.a. Module to control Robotiq's grippers - tested with HAND-E. URL: https://sdurobotics.gitlab.io/ur_rtde/_static/robotiq_gripper.py. Luettu: 13.5.2026.

Robotiq s.a. a. Adaptive Grippers | Robotiq. URL: https://robotiq.com/products/adaptive-grippers. Luettu: 16.5.2026.

Robotiq s.a. b. Wrist Camera | Robotiq. URL: https://robotiq.com/products/wrist-camera. Luettu: 24.5.2026.

Stockfish s.a. Stockfish - Strong open-source chess engine. URL: https://stockfishchess.org/. Luettu: 24.5.2026.

TheRobotStudio s.a. TheRobotStudio/SO-ARM100: Standard Open Arm 100. URL: https://github.com/TheRobotStudio/SO-ARM100. Luettu: 15.4.2026.

Ultralytics 2026a. Model Training with Ultralytics YOLO - Ultralytics YOLO Docs. URL: https://docs.ultralytics.com/modes/train/. Luettu: 27.4.2026.

Ultralytics 2026b. Ultralytics YOLO26 | Ultralytics Docs. URL: https://docs.ultralytics.com/models/yolo26. Luettu: 11.5.2026.

Universal Robots s.a. a. UR16e Collaborative Robot | 16 kg Payload Cobot | Universal Robots. URL: https://www.universal-robots.com/products/ur16e/. Luettu: 24.5.2026.

Universal Robots s.a. b. Real-Time Data Exchange. URL: https://www.universal-robots.com/developer/communication-protocol/rtde/. Luettu: 13.5.2026.

Wilber, J. & Werness, B. s.a. The Importance of Data Splitting. URL: https://mlu-explain.github.io/train-test-validation/. Luettu: 27.4.2026.