

Mehanični Turek

(računalništvo)

Raziskovalna naloga

Mentor: Aleš Volčini, prof.

Avtorji: Aljaž Resman

Jure Grčar

Marko Vidić

G 3.B

Ljubljana, marec 2025

ZAHVALA

Zahvaljujemo se našemu mentorju, Alešu Volčiniju za vloženi čas, nasvete in na splošno za vso pomoč pri izdelavi seminarske naloge in vodstvu šole, ker so nam omogočili dostop in neomejeno uporabo opreme.

POVZETEK

V tej raziskovalni nalogi opisujemo proces izdelave Mehaničnega Turka – programa, ki s spletno kamero in robotsko roko ABB GoFa™ CRB15000 igra šah proti človeškemu nasprotniku. Za razliko od originalnega Turka naša naprava ne zahteva prisotnosti drugega človeka, razen igralca. Projekt je večinoma napisan v programskem jeziku Python, delno pa tudi v jeziku Rapid podjetja ABB. Program preverja stanje na šahovnici s pomočjo ogrodja OpenCV, ugotavlja potezo človeka ter s pomočjo paketa Stockfish odgovori tako, da z robotsko roko naredi svojo potezo.

Ključne besede: ABB GoFa™ CRB15000, Python, šah, OpenCV

KAZALO VSEBINE

1.	UVOD.....	5
2.	RAČUNALNIŠKI VID	7
2.1.	ZAJEMANJE SLIKE.....	7
2.2.	OBDELAVA SLIKE ZA BOLJŠO PREPOZNAVO STANJA NA ŠAHOVNICI	8
2.3.	BARVNE MASKE	8
2.4.	ZAZNAVANJE FIGUR.....	9
2.5.	ZAZNAVANJE ČLOVEŠKE ROKE	10
3.	POTEZE IN ŠAH.....	11
3.1.	IZBIRA ZAPISA.....	11
3.2.	DOLOČANJE POTEZE GLEDE NA ZAJETO SLIKO	12
3.3.	OVOJNICA ZA STOCKFISH.....	12
3.4.	PREPREČEVANJE GOLJUFIJE IN NAPAČNIH POTEZ	12
3.5.	OD POLJA DO KOORDINAT.....	13
4.	VIZUALIZACIJA	14
5.	IZMENJAVA INFORMACIJ	14
5.1.	PROTOKOL IZMENJAVE INFORMACIJ	14
6.	DELOVANJE.....	15
7.	ZAKLJUČEK	16
	VIRI	17
	PRILOGE.....	18

1. UVOD

Za boljše razumevanje te raziskovalne naloge priporočamo znanje programskega jezika Python, seznanjenost z knjižnico OpenCV, osnove poznavanja računalniških omrežji, osnovno razumevanje programskega jezika Rapid, osnovno delovanje in varnostne protokol pri uporabi robotske roke ABB in znanje pravil šaha. Ker so te teme izven domene raziskovalne naloge, jih v raziskovalni nalogi ne bomo dodatno razlagali, razen če bo razlaga pomembno prispevala k razumevanju našega dela.

Šola je z namenom izobraževanja spomladi 2024 v okviru EU projekta NOO dobila robotsko roko ABB GoFa™ CRB15000. Gre za kolaborativno robotsko roko, kar pomeni da je namenjena kontaktu s človekom.

Po začetnem spoznavanju s sistemom, varnostnimi mehanizmi, aplikacijo za upravljanje roke, razvijalskim okoljem ABB RobotStudio® in dokumentacijo, smo iskali konkreten primer uporabe, kjer bi se lahko nekaj naučili in obenem na koncu imeli zanimiv izdelek.

Prišli smo na idejo, da bi sestavili sistem, ki bi vodil robotsko roko pri igranju šaha. Čeprav smo vedeli, da se to da in da se to pojavlja v literaturi izpred 40 let, robota ki bi igral šah še nismo videli in tudi ne poznamo nikogar, ki bi to videl pred nami.

Slišali smo tudi za prevaro iz leta 1770: Predstavili so mehansko napravo, po imenu Mehanični Turek, ki naj bi bila sposobna samostojno igrati šah proti človeškemu nasprotniku. Na Wikipediji celo najdemo podatek, da je premagala samega Napoleona Bonaparta. Kasneje se je izkazalo, da naprava ni bila avtonomna, ampak se je v njej skrival zelo dober igralec šaha. Ime naprave smo zato vzeli za naslov naše raziskovalne naloge.

Nalogo smo razdelili v tri sklope:

- programsko kodo, ki se bo izvajala na krmilniku robotske roke ABB OmniCore™ C30,
- računalniški vid, ki bo s pomočjo kamere zaznaval človeško roko in premike figur in
- kodo, ki bo izmenjala informacije s Stockfish-om in poslala potezo krmilniku robotske roke.

Naš Mehanični Turek zmore:

- zaznati človeško roko,
- zaznati spremembe stanja na šahovnici,
- avtonomno izbrati svojo potezo,
- prpoznavati in popraviti nedovoljene poteze nasprotnika,
- premikati figure,
- izvesti rokado in
- jemati figure.

Česa naš Mehanični Turek še ne zmore:

- en Passant-a in
- promoviranja figur.

Hipoteze naše raziskovalne naloge bi lahko združili v naslednje točke:

- robotsko roko lahko učinkovito upravljamo s programskim jezikom Python,
- s knjižnico OpenCV zanesljivo prepoznavamo položaje figur,
- detektiramo nedovoljene poteze in
- najdemo način, kako odigrati partijo šaha, ne da bi od uporabnika zahtevali karkoli drugega razen premikanja figur.

2. RAČUNALNIŠKI VID

Naš cilj je bil usposobiti Mehaničnega Turka, da avtonomno igra šah. To pomeni, da lahko na poteze človeškega nasprotnika odgovarja samostojno, brez intervencije drugega človeka.

Da bi naš mehanični turek lahko sledil stanju na šahovnici, potrebuje »oči«, kar je v našem primeru spletna kamera. Z njo snemamo stanje na šahovski plošči, s slike, ki jo posreduje, pa moramo razvozlati kakšno to stanje je.

Knjižnica OpenCV je popularna knjižnica za računalniški vid, napisana za vrsto različnih programskih jezikov. Zanj smo se odločili, ker smo pred tem to knjižnico že uporabljali drugje in smo si že nabrali nekaj predznanja. Knjižnica vsebuje veliko pomembnih algoritmov za barvne maske in odpravljanje šuma v njih.

Uporaba knjižnice OpenCV je za sabo potegnila tudi odločitev o izbiri programskega jezika. Za programski jezik Python smo se odločili predvsem zaradi dobre integracije te knjižnice in poenostavljene uporabe. Python nam je ustrezal tudi zaradi preproste komunikacije s krmilnikom, ki robota upravlja.

V nadaljevanju bomo opisali kako obdelamo sliko, ki jo zajamemo s kamero, ki je nameščena na prijemalniku roke.



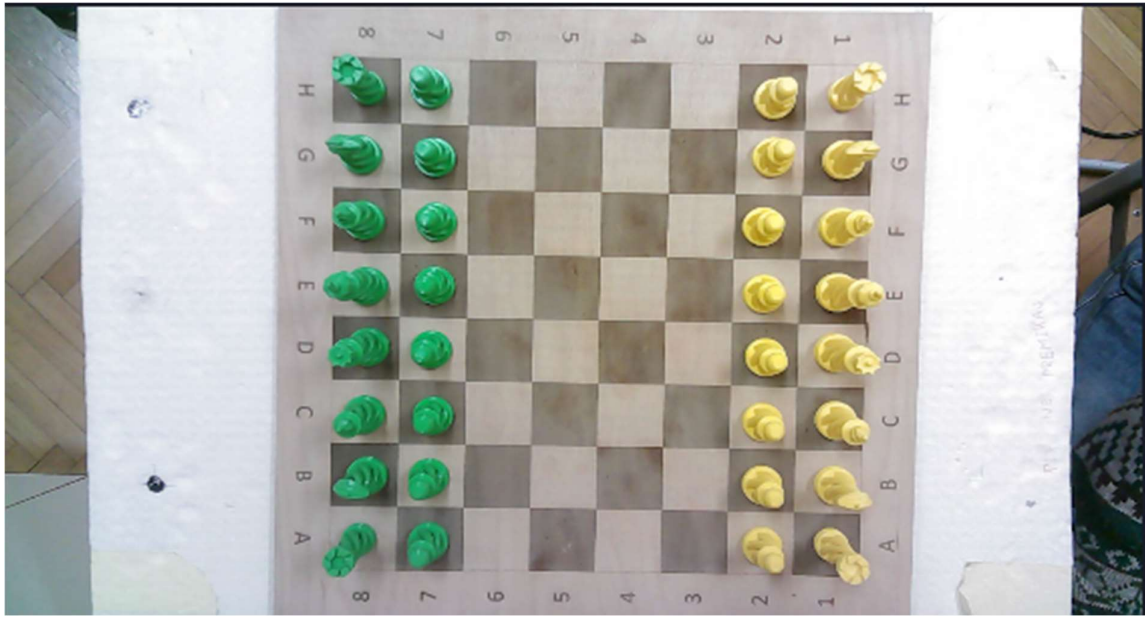
Slika 1: Pritrditev kamere na robotsko roko

2.1. ZAJEMANJE SLIKE

Sliko zajamemo s pomočjo knjižnice OpenCV. Ta knjižnica ima vgrajeno metodo:

```
cv2.VideoCapture(num: int).read() -> (return_value:int, image: int64),
```

ki kot argument vzame število, s katerim se odloči katero kamero bo uporabila, vrne pa status in zajeto sliko.



Slika 2: Zajeta slika šahovnice pred obdelavo

2.2. OBDELAVA SLIKE ZA BOLJŠO PREPOZNAVO STANJA NA ŠAHOVNICI

Ker je kamera zaradi pričvrstitve na robotsko roko glede na šahovnico zarotirana 90° , za lažje delo sliko prvo programsko obrnemo. Sliko še obrežemo, da se izognemo napačnim zaznavam barve.

2.3. BARVNE MASKE

Ko je slika obrezana in obrnjena, začnemo z zaznavanjem figur. Odločili smo se za način zaznavanja figur s pomočjo zoženja barvnega razpona. Ta način se nam je zdel najboljši, ker zanj ni potrebno veliko procesorske moči. Potrebovali pa smo figure primernih barv. Te smo 3D natisnili v temno zeleni in svetlo rumeni barvi, ker sta ti dve barvi dovolj kontrastni glede na šahovnico.

Barvno masko smo določili glede na razpon barv, ki smo ga za temne in svetle figure določili z eksperimentiranjem in merjenjem RGB vrednosti. Barvni razpon smo izbrali tako, da je barve mogoče zaznati pri različnih jakostih osvetlitve.

```
yellow_mask = cv2.inRange(frame, lowerb=np.array([91, 188, 210]), upperb=np.array([166, 247, 235]))
```

Sprva je bilo zaznavanje barve težava, saj so v barvni maski pogosto nastopili šumi. Tem smo se izognili s pomočjo popačenja (ang. dilation). Popačenje deluje glede na v naprej določeno jedro (ang. kernel), kar je matrika prižganih bitov. Velikost in oblika te matrike vplivata na način in smer popačenja. Obe lastnosti jedra smo določili s poskušanjem različnih vrednosti.

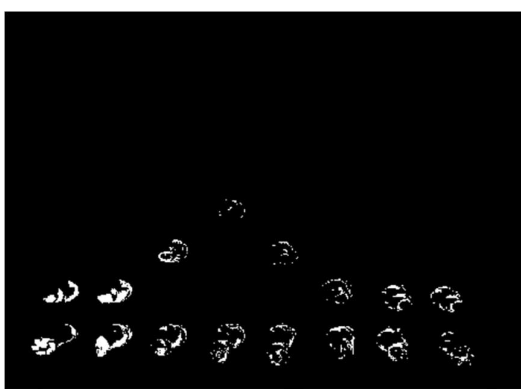
```
kernel = np.ones((5, 5), np.uint8) # kernel z obliko (5, 5)
```

$$\text{dst}(x, y) = \max(x', y') : \text{element}(x', y') \neq \text{osrc}(x + x', y + y')$$

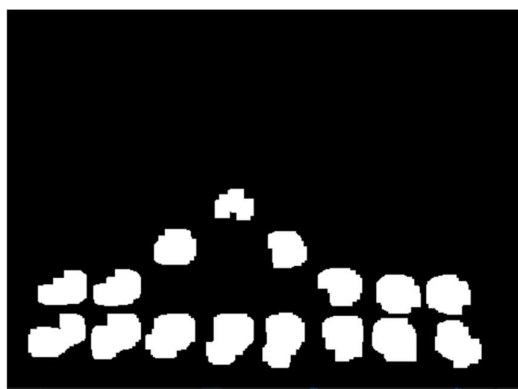
Slika 3: Formula popačenja ki ga uporabljamo

Zajeto sliko popačimo tako, da izstopajoča mesta ojačamo do te mere, da tvorijo na sliki nek večinoma konveksen poligon, ki ustreza položaju figur ena šahovski plošči. Za popačenje uporabimo vgrajeno funkcijo:

```
yellow_mask = cv2.dilate(yellow_mask, kernel, iterations = 3)
```



Slika 4: Rumena maska pred popačenjem



Slika 5: Rumena maska po popačenju

2.4. ZAZNAVANJE FIGUR

S pomočjo barvnih mask določimo na katerih poljih so figure (slika 6 in slika 7) in katere barve so.

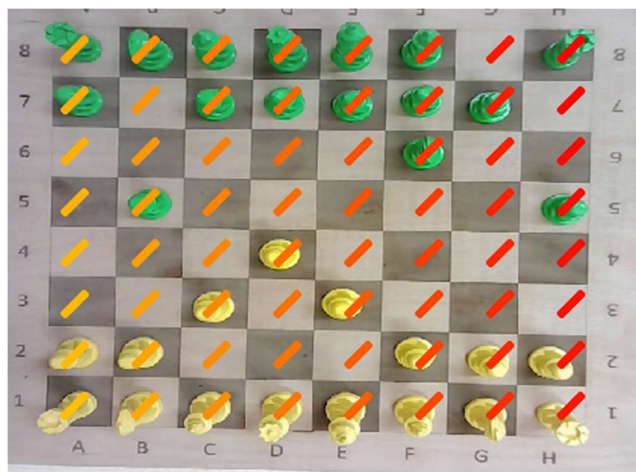


Slika 6: Naključno stanje na šahovnici



Slika 7: Maska rumenih figur za dano stanje

Pozicije na sliki, kjer iščemo figure so nastavljene v naprej. Na vsaki poziciji ne preverjamo samo ene točke temveč sklop točk na način, kot ga vidimo na sliki 8. Stanje zapišemo v niz znakov, ki imitira FEN šahovsko notacijo, ki je razložena v nadaljevanju članka.



Slika 8: Detekcijske točke, kjer mehanski turek išče figure.

Naša notacija za vsako polje posebej pove ali je na njem svetla figura – 'w', temna figura – 'b', ali pa je polje prazno – '0'. Primer zapisa za stanje na šahovnici na sliki 8:

```
bbbbbb0b/b0bbbb0/0000b00/0b0000b/000w0000/00w0w000/ww000www/wwwwwwwww/
```

Opisan proces je realiziran v svoji metodi »get_fen_from_pic« v datoteki pogled.py.

2.5. ZAZNAVANJE ČLOVEŠKE ROKE

Zaznavanje človeške roke je nujno potrebno, zato, da z robotsko roko ne posegamo v območje šahovnice medtem ko se človek odloča o potezi. S tem se izognemo paniki, ki bi lahko nastala, če bi se robotska roka nenapovedano premaknila ali se celo dotaknila človeka.

Zaznavanje rok je implementirano v razredu »Roke«, ki je del datoteke RokeActions.py. Razred »Roke« ima dve razredni metodi:

- getStatus – metoda, ki vrne trenutno stanje spremenljivke STATUS, ki vsebuje informacijo o prisotnosti človeške roke v zadnjih treh sekundah in
- updateRoke – metoda, ki jo kličemo v posebni niti, enkrat na sekundo. Metoda preveri prisotnost človeške roke in v primeru, da je človeška roka odsotna 3 sekunde nastavi vrednost STATUS-u.

3. POTEZE IN ŠAH

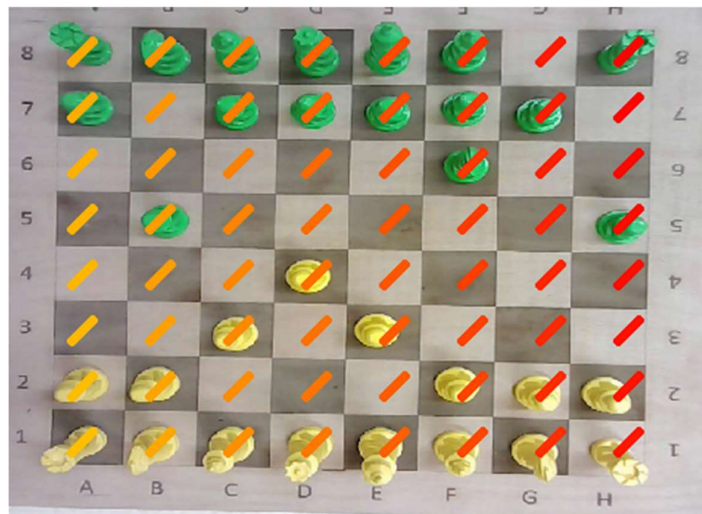
Za izračun naslednje najboljše poteze smo uporabili Stockfish. To je svetovno znan odprtokodni pogon za šah, ki je osvojil naslov svetovnega prvaka v šahu med računalniškimi programi in ki ga drži še danes. Izbrali smo ga, ker je brezplačen in ima odličnega vmesnika za Python. Ta pride v obliki knjižnice z istim imenom in s svojimi metodami omogoča enostavno komunikacijo s pogonom, ki je ključnega pomena za projekt.

3.1. IZBIRA ZAPISA

V svetu šaha se uporabljajo trije glavni zapisi:

- Algebraična notacija – Zapis, ki nam pove katera figura se je premaknila kam
- Univerzalni šahovski vmesnik (UCI) – Način zapisa, ki pove iz katerega polja in na katero polje smo nekaj premaknili.
- Forsyth-Edwards Notacija (FEN) – Način zapisa, ki opiše celotno stanje na šahovnici.

Naš Mehanični Turek uporablja UCI in FEN notaciji. Najbolj primerna se nam je zdela notacija FEN, saj poleg stanja na šahovnici nosi še informacije o tem kdo je na potezi in ali sme narediti rošado.



Slika 9: Naključna postavitev figur

Primer FEN notacije za zgornjo sliko:

```
rnbqkb1r/p1ppppp1/5n2/1p5p/3P4/2P1P3/PP3PPP/RNBQKBNR w KQkq - 1 4
```

Ta notacija opiše celotno stanje na šahovnici, kjer so vrstice ločene z poševnicami. Črne figure (naše zelene) so označene z malimi črkami, bele (naše rumene) pa z velikimi.

3.2. DOLOČANJE POTEZE GLEDE NA ZAJETO SLIKO

Ko potrdimo, da je uporabnik naredil potezo in imamo vse pozicije črnih in belih figur, moramo identificirati figure na posameznih poljih. Tega postopka smo se lotili z metodo »get_fen«. Ta metoda kot argumente vzame prejšnje stanje na šahovnici in naš zapis pozicij črnih in belih figur. Metoda prešteje število figur in ga primerja s številom v prejšnjem zapisu. S pomočjo »if« stavkov preveri različne možnosti kot na primer jemanje figur in rokade. S tem določi novo stanje na šahovnici. Metoda vrne nov FEN. S pomočjo metode »get_uci« na podlagi prejšnjega in zdajšnjega FEN določimo potezo v UCI formatu, ki jo dodamo na seznam odigranih potez. Ta seznam posredujemo Stockfish-u, ki na njegovi podlagi določi naslednjo potezo. Tudi to potezo dodamo v seznam odigranih potez.

3.3. OVOJNICA ZA STOCKFISH

Da bi kodo skrajšali in jo naredili bolj berljivo, smo napisali ovojnico za Stockfish. Ta združi vse tri metode, ki jih potrebujemo za generiranje poteze. Najprej definiramo pot do datoteke, ki vsebuje pogon Stockfish. Nato inicializiramo razred Stockfish s parametri o nivoju sposobnosti, s katero bo naš Mehanični Turek igral. Datoteka stockfish_wrapper.py vsebuje metodo get_move, ki potrebuje argument seznama potez v UCI notaciji, vrne pa »najboljšo« potezo in stanje na šahovnici po izvedeni potezi.

```
def get_move(fen:str):
    stockfish.set_fen_position(fen)
    move = stockfish.get_best_move() #stockfish najde najboljšo potezo
    stockfish.make_moves_from_current_position([move]) #posodobi FEN

    return move, stockfish.get_fen_position() #stockfish nam vrne potezo in FEN
```

3.4. PREPREČEVANJE GOLJUFIJE IN NAPAČNIH POTEZ

Ko uporabnik naredi potezo, in jo kamera zazna, moramo preveriti legalnost te poteze. To storimo s pomočjo knjižnice Stockfish. Tako izvemo ali je legalna ali ni. Če premik figure ni legalen, jo Mehanični Turek premakne nazaj. Tak način ima nekaj slabosti in sicer v primeru požiranja, kjer mora človeški igralec na polje vrniti figuro, ki jo je požrl.

```
def is_move_legal(prev_fen: str, new_fen: str) -> bool:
    """Validate moves using python-chess."""
    board = chess.Board(prev_fen)
    uci_move = get_uci(prev_fen, new_fen)
    if not uci_move: # če ni bilo poteze, vrnemo false
        return False
    move = chess.Move.from_uci(uci_move)
    return move in board.legal_moves # vrni ali je poteza legalna
```

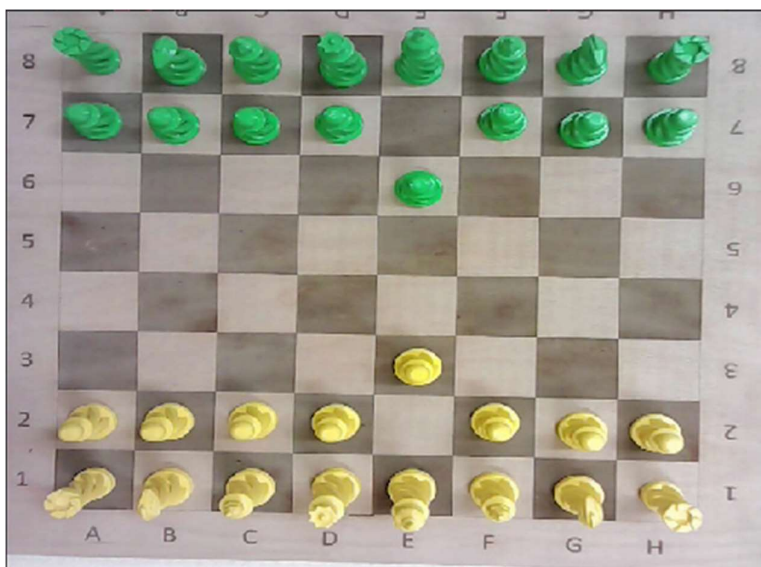
3.5. OD POLJA DO KOORDINAT

Stockfish vrne potezo in sicer v UCI notaciji. Ker želimo to potezo uporabiti za premik robotske roke, jo je potrebno pretvoriti v dva sklopa koordinat v resničnem svetu, ki jih krmlinik robotske roke razume. Ta dva sklopa koordinat sta pozicija, kjer se figura trenutno nahaja, in pozicija, kamor jo želimo premakniti. Obe poziciji vključujeta tudi višino figure, ki jo je potrebno pobrati, saj se naše šahovske figure močno razlikujejo po višini. Višine figur so predefinirane za vsako figure posebej.

Za pretvorbo poteze v koordinate smo pripravili naslednje funkcije:

- `get_coords` – pomaga pri izračunu koordinat vsakega polja,
- `check_square` – preveri ali je na polju že kakšna figura,
- `piece_height` – preveri katera figura je na polju in vrne njeno višino in
- `katera_figura` – pove katera figura je na danem polju.

Izjema pri pretvorbi sta rošada in jemanje figur. V obeh primerih je potrebno premakniti dve figuri, za kar potrebujemo štiri sklope koordinat. Dobimo jih tako, da metodo za določanje koordinat izvedemo dvakrat, za vsako figuro posebej.

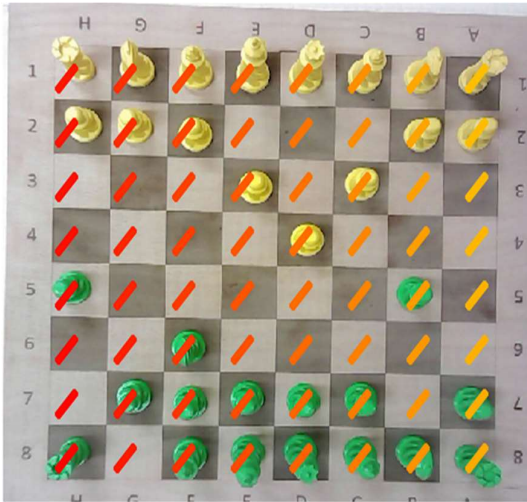


Slika 10: Stanje na šahovnici in pretvorba koordinat

Primer: Ko ugotovimo pozicijo kmeta na E6 in jo preračunamo v koordinate robotske roke, pridemo do položaja (200mm, 250mm) od izhodišča na polju A1.

4. VIZUALIZACIJA

Da bi Mehaničnega Turka naredili lažje predvidljivega, smo napisali preprosto grafično vizualizacijo. Deluje z uporabo knjižnice PyGame, ki jo uporabljamo za prikazovanje slik v oknu (slika 11 in slika 12). Deluje tako, da premika slike figur okrog po sliki šahovnice. To smo dosegli z metodo »see_board«, ki kot argument vzame FEN šahovnice.



Slika 11: Naključno stanje na šahovnici



Slika 12: Vizualizacija stanja na sliki 11

5. IZMENJAVA INFORMACIJ

ABB Rapid je privzeti programski jezik, ki se izvaja na krmilniku robotske roke. Ker naša koda deluje v Pythonu, in je ne moremo izvajati na krmilniku same roke, jo izvajamo na ločenem računalniku in krmilniku posredujemo potrebna dejanja, zato se moramo na krmilnik tudi povezati preko računalniškega omrežja in protokola TCP/IP.

5.1. PROTOKOL IZMENJAVE INFORMACIJ

Ker sta računalnik, na katerem teče naš program in krmilnik robotske roke povezana v omrežje ethernet, morata biti ethernet vmesnika nastavljena tako, da sta napravi v skupnem omrežju. Za lažje delo smo obema smo določili statična IP-ja in vrata skozi katera bo potekala komunikacija.

Naša koda za krmilnik roke ki je napisana v Rapid-u se poveže s pomočjo TCP/IP protokola:

```
VAR string SERVER_IP:="192.168.125.123";  
VAR num SERVER_PORT:=65432;  
VAR socketdev client_socket;  
  
SocketCreate client_socket;  
SocketConnect client_socket, SERVER_IP, SERVER_PORT;
```

Ko sta povezana, krmilnik sprejme koordinate ki jih določimo po prej opisanem procesu.

```
SocketReceive client_socket \Str:=coords1X;  
SocketReceive client_socket \Str:=coords1Y;  
SocketReceive client_socket \Str:=coords1Z;  
SocketReceive client_socket \Str:=coords2X;  
SocketReceive client_socket \Str:=coords2Y;  
SocketReceive client_socket \Str:=coords2Z;
```

Posamične vrednosti koordinat nato pretvorimo v krmilniku razumljiv format, s katerim robota pošljemo na željeno lokacijo.

```
SockLoc2.trans.x:= SockLoc2.trans.x + coords1X1;  
SockLoc2.trans.y:=SockLoc2.trans.y + coords1Y1;  
GRAB2.trans.z:=GRAB2.trans.z + coords1Z1;  
MoveL SockLoc2, v400, fine, tool0;
```

Po izvedeni potezi vedno vrne sporočilo Python procesu za potrditev izvedbe poteze.

```
SocketSend client_socket\Str:="move";
```

To ponavlja v neskončnost, dokler Python proces ne prekine povezave preko internetnega socketa.

6. DELOVANJE

Naš Mehanični Turek je sposoben odigrati celo partijo šaha. Najprej računalnik in krmilnik robotske roke vzpostavita TCP/IP povezavo. Ko je robotska roka pripravljena na potezo pošlje računalniku niz »move«. Tako računalnik ve, da lahko zajame sliko šahovnice in z algoritmi opisanimi v prejšnjih poglavjih določi trenutno stanje na šahovnici. Stanje, ki ga dobi z maskami spremeni v točno stanje vseh figur in ga zapiše v FEN notaciji. Stockfish-u posreduje FEN notacijo. Preveri ali je poteza legalna in če je, nam vrne najboljšo potezo. Potezo spremenimo v fizične koordinate na šahovnici in jih preko TCP/IP posredujemo krmilniku robotske roke. Robotska roka se premakne da naredi potezo. Ko je te poteze konec se robotska roka vrne nazaj na pozicijo za slikanje šahovnice. Preko TCP/IP spet pošlje »move«. Celoten proces se ponovi.

7. ZAKLJUČEK

Naš Mehanični Turek dosega vse cilje, ki smo si jih zastavili. Čas med potezami je sorazmerno nizek, a bi lahko bil z nadgradnjo fizične opreme še nižji. Mehanični Turek zanesljivo igra šah proti človeškemu igralcu na precej visokem sposobnostnem nivoju. Koda je zaradi večje preglednosti smiselno razdeljena v datoteke. To olajša tudi prihodnje izboljšave.

V prihodnosti bi si želeli izboljšati predvsem zanesljivost projekta. V trenutnem stanju se včasih proti koncu igre zgodi napaka, ker Mehanični Turek želi narediti rošado, vendar te zmožnosti nima. Program se ustavi in potreben je ponovni zagon.

S svojim projektom smo zelo zadovoljni, in ga bomo v prihodnosti zagotovo še izpopolnjevali. Načrtujemo razvoj možnosti promoviranja figur in avtonomne priprave šahovnice za začetek igre.

Uresničile so se vse naše hipoteze. Naš mehanični Turek lahko povsem avtonomno prepozna stanje na šahovnici in odigra svojo potezo. Lahko tudi prepozna in popravi nasprotnikove nepravilne poteze.

VIRI

- [1] OpenCV: Eroding and Dilating, pridobljeno 6. 3. 2025, dosegljivo na https://docs.opencv.org/3.4/db/df6/tutorial_erosion_dilatation.html
- [2] OpenCV-Python Tutorials, pridobljeno 6. 3. 2025, dosegljivo na https://docs.opencv.org/4.x/d6/d00/tutorial_py_root.html
- [3] Stockfish repozitorij, preneseno 6. 3. 2025, dosegljivo na <https://github.com/official-stockfish/Stockfish>
- [4] Mechanical Turk, pridobljeno 6. 3. 2025, dosegljivo na https://en.wikipedia.org/wiki/Mechanical_Turk
- [5] Ilya Zhelyabuzhsky: Stockfish python knjižnica, preneseno 6. 3. 2025, dosegljivo na <https://pypi.org/project/stockfish/>
- [6] PyGame dokumentacija, pridobljeno 6. 3. 2025, dosegljivo na <https://www.pygame.org/docs/>
- [7] Forsyth-Edwards Notation (FEN), pridobljeno 6. 3. 2025, dosegljivo na <https://www.chess.com/terms/fen-chess>
- [8] ABB: Technical reference manual, preneseno 6. 3. 2025, dosegljivo na https://library.e.abb.com/public/b227fcd260204c4dbeb8a58f8002fe64/Rapid_instructions.pdf
- [9] Forsyth-Edwards Notation, pridobljeno 6. 3. 2025, dosegljivo na https://en.wikipedia.org/wiki/Forsyth%E2%80%93Edwards_Notation

PRILOGE

- Priloga 1: Repozitorij s programsko kodo:
<https://github.com/VegovaABB/Chess>