



UNIVERSITATEA DIN BUCUREŞTI



FACULTATEA  
DE  
MATEMATICĂ ȘI INFORMATICĂ

SPECIALIZAREA INFORMATICĂ

**Lucrare de licență**

**MINI BRAT ROBOTIC  
PENTRU PONG**

Absolvent

**Dilirici Radu**

Coordonatori științifici

**Conf. Dr. Bogdan Alexe**

**Coord. laborator robotică Andrei Dumitriu**

București, iunie 2021

## Abstract

Lucrarea curentă tratează subiectul roboților autonomi, având un caracter recreativ. Mai precis, constă într-un mini braț robotic care reușește să joace Pong, un joc video clasic, fără intervenția utilizatorului.

Jocul este rulat pe un Raspberry Pi și afișat pe un ecran în poziție orizontală. Paletele jucătorilor sunt controlate prin două joystick-uri (una pentru fiecare). Robotul este complet separat de mașina pe care rulează jocul și observă stadiul jocului prin intermediul unei camere video. Imaginele sunt procesate, iar datele importante sunt extrase datele din acestea. Pe baza acestor informații, robotul decide în ce direcție să miște joystick-ul lui, practic simulând o altă persoană chiar și la nivel fizic.

Pentru luarea acestor decizii au fost încercate diferite metode, precum descrierea completă a comportamentului robotului, sau antrenarea unor agenți de Reinforcement Learning. Cele mai bune rezultate au fost obținute de algoritmul determinist, căruia i-au fost scrise instrucțiunile exacte de funcționare, iar performanța acestuia este peste media jucătorilor umani.

Componentele principale ale robotului sunt un Raspberry Pi (diferit de cel pe care rulează jocul) care se ocupă de procesarea datelor și aplicarea algoritmului, o cameră RGB responsabilă pentru preluarea datelor și un servomotor care mișcă unul dintre joystick-uri.

Pentru că programul rulează pe o componentă hardware separată, el este independent de platforma jocului. Astfel, ar putea juca, de exemplu, jocuri pe un PC cu Windows, un PlayStation 2, un telefon cu iOS, sau chiar și pe o stație arcade, chiar dacă nu toate suportă rularea unui agent de IA pe ele.

# Abstract

This paper deals with the subject of autonomous robots, having a recreational character. More precisely, it consists of a mini robotic arm that manages to play Pong, a classic video game, without user intervention.

The game runs on a Raspberry Pi and is displayed on a screen in a horizontal position. Players' palettes are controlled by two joysticks (one for each). The robot is completely separated from the machine on which the game is running and observes the game stage via a video camera. The images are processed and the relevant data is extracted. Based on the new information, it decides in which direction to move its joystick, practically simulating another person even on a physical level.

Through the project, different methods have been tried to make these decisions, such as the thorough description of the robot's behavior, or the training of Reinforcement Learning agents. The best results were obtained by the deterministic algorithm, to which the exact operating instructions were written, and its performance is above the average of the human players.

The main components of the robot are a Raspberry Pi (different from the one running the game) that deals with data processing and the application of the algorithm, an RGB camera responsible for getting the data and a servo motor that moves one of the joysticks.

Because it runs on a separate hardware component, it is independent of the game platform. Thus, it could play, for example, games on a Windows PC, a PlayStation 2, an iOS phone, or even on an arcade station, even if not all of them support running an AI agent on them.

# Cuprins

<b>Abstract</b>	<b>2</b>
<b>Abstract</b>	<b>3</b>
<b>Cuprins</b>	<b>4</b>
<b>Introducere</b>	<b>4</b>
<b>I. Descrierea jocului Pong</b>	<b>8</b>
I.1 Hardware	8
I.2 Software	10
<b>II. Brațul robotic autonom</b>	<b>14</b>
II.1 Versiuni anterioare	14
II.2 Hardware	15
II.3 Procesarea datelor	17
II.4 Algoritmul determinist	19
II.5 Reinforcement Learning	23
II.5.1 Deep Q-Learning	23
II.5.2 Rezultate și Experimente	25
II.5.3 Folosirea coordonatelor	26
<b>III. Rezultate și Concluzii</b>	<b>28</b>
<b>Bibliografie</b>	<b>30</b>
<b>Listă figuri</b>	<b>31</b>

# Introducere

Lucrarea aparține atât domeniului robotic, cât și al Inteligenței Artificiale. Această combinație câștigă popularitate pe zi ce trece. Există tot mai multe produse care reușesc să îmbine aceste concepte precum roboți care livrează colete singuri, roboți de securitate care patrulează o zonă, sau mașinile autonome, una dintre cele mai răspândite idei actuale. Lucrarea de licență ia în considerare această creștere a interesului în robotica intelligentă și abordează o temă din acest domeniu.

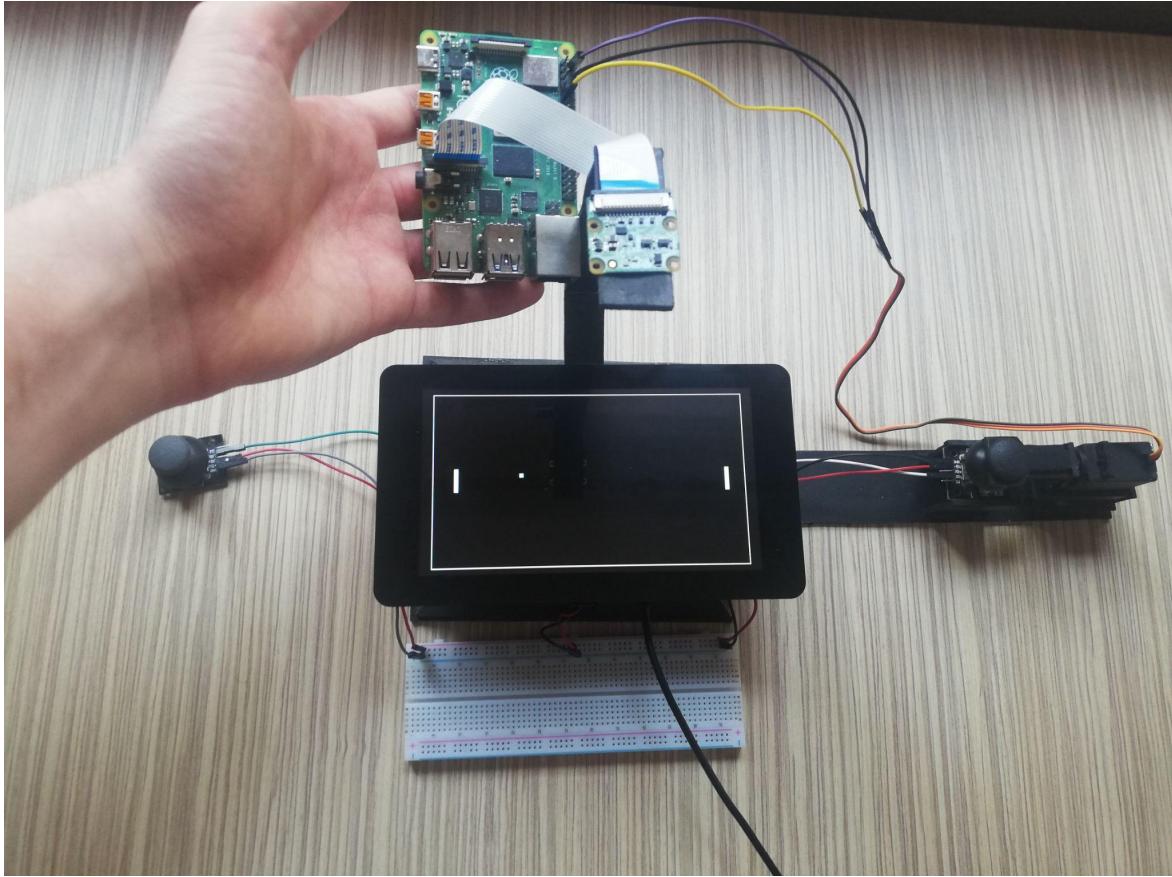
Am ales acest proiect datorită experienței acumulate în crearea jocurilor video. Astfel, realizarea jocului a necesitat un efort relativ scăzut, iar accentul a căzut pe implementarea robotului și a agentului.

De asemenea, de multe ori în acest domeniu se pune problema eticii și a siguranței, cum se poate observa în cazul mașinilor autonome. Cum acest proiect este unul inofensiv - un robot care poate juca un joc - aceste lucruri nu au trebuit să fie luate în considerare, oferind un avantaj în cazul lansării produsului.

În același timp, am observat că jucarea jocurilor video "din exterior" ce către roboți este un domeniu foarte slab dezvoltat. Când ne referim la Inteligență Artificială în aria jocurilor video, în general ne gândim la algoritmi care sunt în legătură directă cu jocul și rulează pe aceeași mașină cu acesta. Lucrarea explorează rularea agentului pe un dispozitiv diferit.

Un alt factor important pentru alegerea temei a fost potențialul acesteia. Jocul Pong este o simplificare a tenisului de masă. Proiectul demonstrează fezabilitatea unui robot care poate juca Ping-Pong cu ajutorul tehniciilor folosite. Un astfel de robot se poate folosi pentru antrenarea sportivului.

În continuare, sistemul ar putea fi îmbunătățit să se descurce cu o varietate mai mare de jocuri, făcându-l un agent universal.



### *Varianta finală a proiectului*

Lucrarea de licență este compusă din trei părți, și anume **Descrierea jocului Pong, Brațul robotic autonom și Rezultate și concluzii.**

În primul capitol este prezentat jocul Pong, atât la nivel hardware, cât și la nivel software. Se vor preciza componentele utilizate, cum au fost acestea asamblate și ce conexiuni au fost realizate pentru funcționarea sistemului. De asemenea, se va explica cum a fost implementat și apoi optimizat jocul pentru a rula la 60 de cadre pe secundă.

Cel de-al doilea capitol constă în prezentarea robotului inteligent. Vor fi descrise componentele folosite și se va preciza cum au fost acestea asamblate și poziționate pentru a observa ecranul jocului. Apoi, se va explica cum datele preluate de camera RGB sunt procesate și, astfel, sunt aflate coordonatele obiectelor din joc. După aceea, se vor prezenta cele două strategii principale care au fost abordate pentru crearea robotului intelligent.

Prima metodă reprezintă un algoritm determinist și se bazează pe cunoașterea în amănunt a proprietăților jocului. Cea de-a doua metodă constă în antrenarea unui agent

inteligent de Reinforcement Learning. Se vor preciza procesul de dezvoltare și experimentele realizate pentru ambele abordări.

În final se vor preciza toate rezultatele obținute și ce concluzii au fost trase pe baza acestora.

# I. Descrierea jocului Pong

Pong este un joc video clasic și unul dintre primele jocuri de tip arcade. Acesta se joacă în două persoane și simulează tenisul de masă.



*Figura 1.1: Jocul Pong*

În acest capitol vom prezenta asamblarea componentelor hardware și programarea jocului. Vor fi descrise componentele sistemului, carcasa confectionată și conexiunile realizate. Pe plan software se vor explica: implementarea jocului, optimizarea acestuia pentru a rula la 60 de cadre pe secundă și tehnologiile folosite.

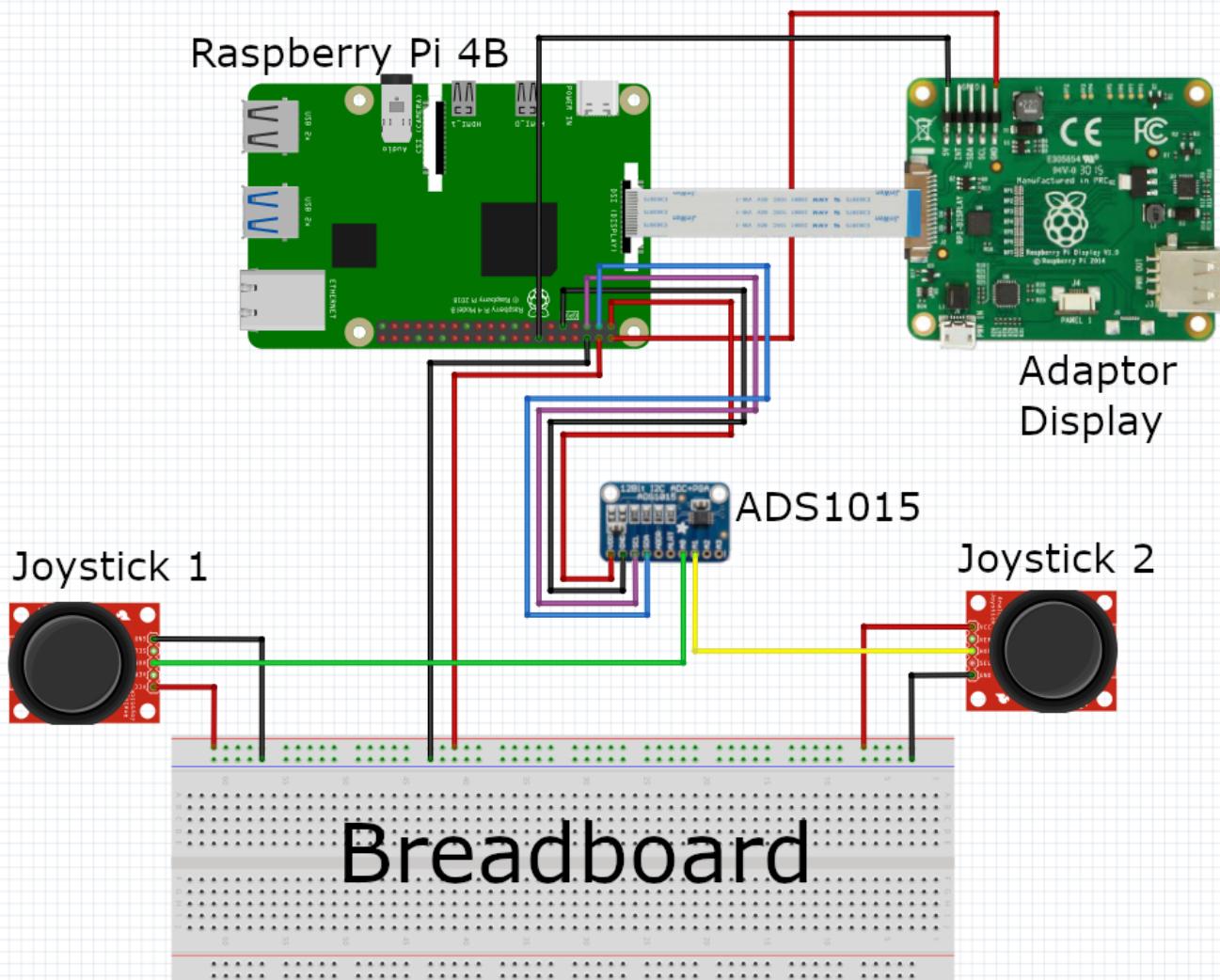
## I.1 Hardware

Jocul este rulat pe un Raspberry Pi 4B, un mini calculator căruia îl se pot conecta diferite componente și senzori. Acesta are o putere computațională suficientă pentru rularea unui joc și are dimensiuni foarte mici. De asemenea, oferă o flexibilitate mare cu privire la limbajul de programare.

Raspberry Pi-ul este conectat la un ecran cu diagonala de 7 inch și touchscreen. Datorită mărimii sale, acesta oferă o experiență plăcută jucătorului. În același timp, funcția de touchscreen s-a dovedit a fi foarte utilă, deoarece, după scrierea codului, jocul se poate rula foarte ușor prin intermediul ecranului.

De asemenea, pentru a controla cele două palete sunt folosite două joystick-uri analog. Pentru că Raspberry Pi suportă doar inputuri de tip digital, s-a utilizat un convertor analog - digital *Adafruit ADS1015* [8]. Pentru alimentarea joystick-urilor și conectarea acestora cu convertorul a fost folosit un breadboard.

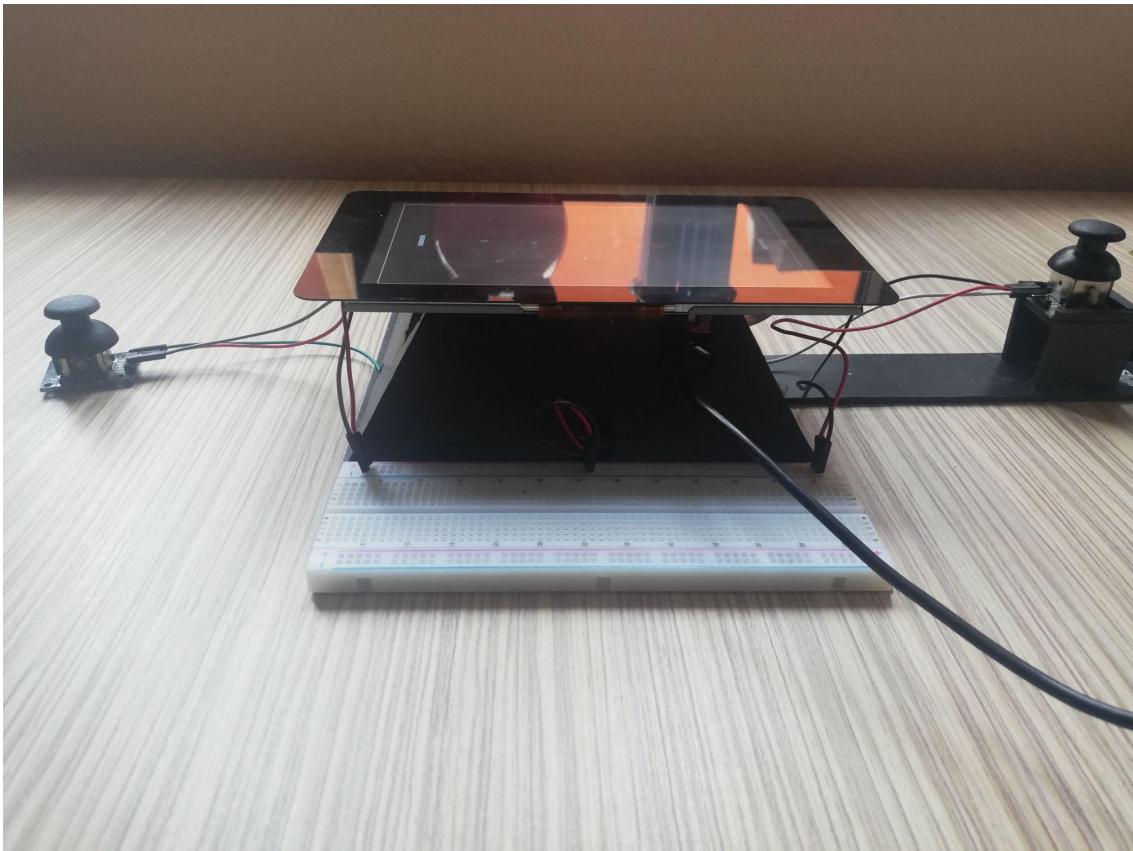
Alimentarea mini calculatorului se realizează cu ajutorul unității de alimentare oferită de Raspberry Pi. Toate celelalte componente sunt alimentate prin mini calculator. În figura 1.2 se pot observa conexiunile făcute.



**Figura 1.2: Sistemul pentru Pong**

*Ecranul este integrat cu adaptorul pentru display*

Am făcut o carcăsă din hobbyglas negru pentru a susține ecranul într-o poziție orizontală și pentru a acoperi cablarea. De asemenea, joystick-ul robotului este fixat printr-un suport.

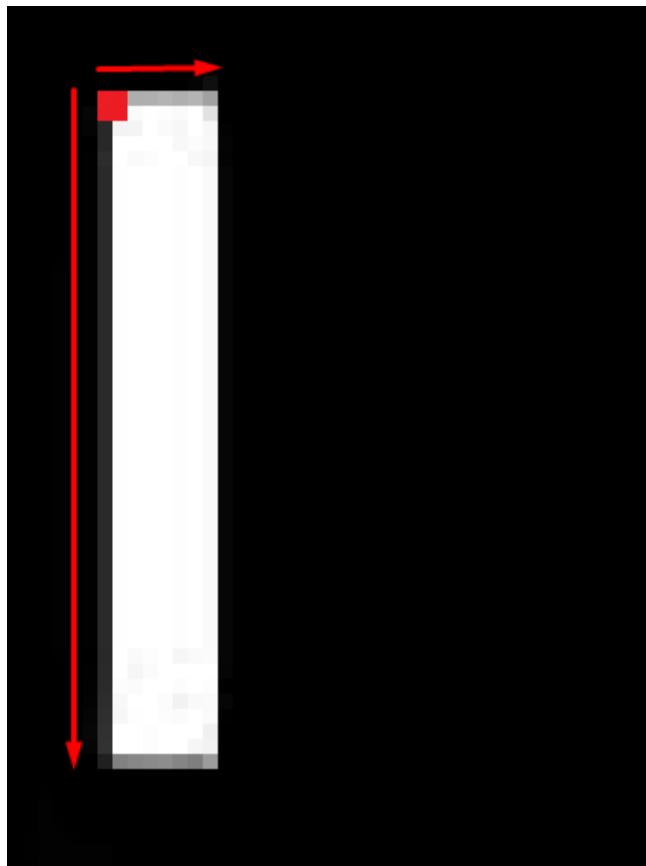


**Figura 1.3: Carcasa jocului**

## I.2 Software

Jocul Pong a fost programat folosind limbajul Python. Pentru afișare a fost folosită librăria *PyGame* [7], iar pentru citirea valorilor joystick-urilor librăria *Adafruit\_ADS1015* [1]. Toate obiectele (cele două palete și mingea) au un punct “de pornire”, acesta fiind colțul lor din stânga sus. De asemenea, sunt reținute și folosite înălțimile și lățimile obiectelor. Paletele au coordonate fixe și prestatabile pe axa orizontală. Inițial, ele se află la mijlocul ecranului (pe

axa verticală), iar mingea se află puțin mai sus de centrul ecranului. Direcția de pornire a mingii este mereu spre jucătorul din stânga (cel uman).



*Figura 1.4: Datele care descriu obiectele*

*Se poate observa punctul “de pornire” și cele două dimensiuni reținute*

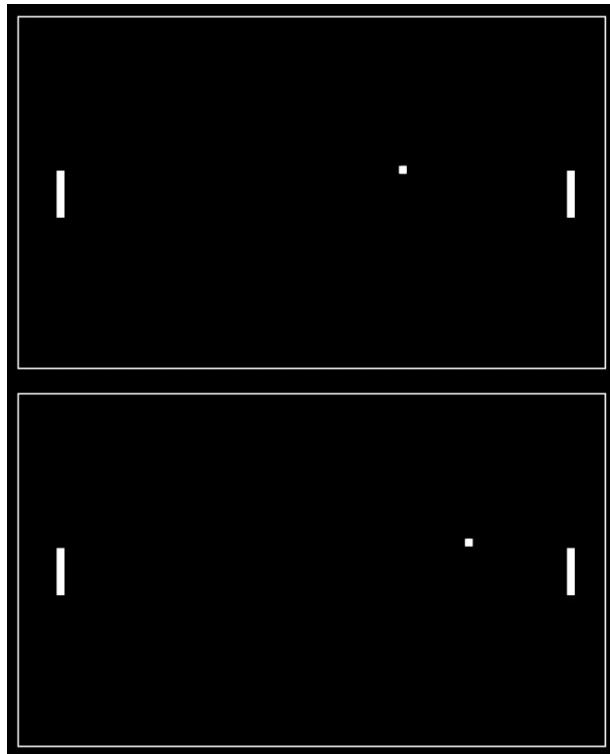
Logica jocului este una simplă. Din păcate, regulile oficiale ale jocului nu au fost publicate, aşa că am încercat să îl recreem pe baza observațiilor. Dacă bila este lovită de o paletă, ea ricoșează, iar unghiul depinde strict de ce porțiune a paletei a atins mingea; astfel, dacă mingea este lovită cu centrul paletei, aceasta se va mișca doar pe axa orizontală, indiferent de unghiul la care a lovit aceasta paleta. Cu cât mingea este lovită mai departe de centrul paletei, cu atât unghiul față de axa orizontală este mai mare. Viteza pe axa orizontală este mereu aceeași. Din acest motiv, o bilă cu traекторie oblică se va mișca mai rapid decât una cu traекторie orizontală (dar va parurge drumul dintre palete în același timp). Minge ricoșează dacă atinge un perete. Dacă trece de o paletă, înseamnă ca jucătorul advers a câștigat runda. Viteza bilei crește cu fiecare lovitură consecutivă, până la o valoare

prestabilită și este resetată la înscrierea unui gol. În cazul acesta, viteza crește de la 8 unități pe secundă până la 12 unități pe secundă.

Controlul paletelor se realizează cu ajutorul unor joystick-uri. Considerăm unghiul  $0$  reprezentat de punctul de repaus al acestora. Dacă unghiul aparține intervalului  $[-\alpha, \alpha]$ , unde  $\alpha$  este o valoare arbitrară, lăsăm paleta în poziția curentă. Dacă unghiul este mai mare decât  $\alpha$  / mai mic decât  $-\alpha$ , mutăm paleta în direcția corespunzătoare.

În realitate, în loc de un unghi primim o valoare numerică pozitivă din intervalul  $0-2500$  (aproximativ), însă aplicăm același principiu. Joystick-ul folosește două potențiometre (câte unul pentru fiecare axă), iar valoarea rezultată e direct proporțională cu voltajul curentului care circulă prin el. Pentru joc folosim doar valorile axei verticale. Valoarea  $0$  ar însemna că joystick-ul este înclinat în jos, iar  $2500$  ar însemna că este în sus. Când acesta este în poziția de repaus, valoarea citită ar trebui să fie  $1250$ . Paleta este mișcată în sus când valoarea citită este peste  $1500$  și în jos când valoarea este sub  $1000$ .

Pe lângă cele trei obiecte precizate sunt afișate și marginile ringului (acesta nu ocupă toată suprafața ecranului) pentru a face detecția ecranului mult mai ușoară.



*Figura 1.5: Jocul Pong*

*Se poate observa mișcarea bilei în cele două cadre*

Chiar dacă jocul este unul simplu, acesta rula la mai puțin de 40 de cadre pe secundă. Am observat că programul, chiar dacă nu rula destul de rapid, folosea maxim 25% din puterea procesorului. Din acest motiv am împărțit tot procesul în trei thread-uri. Primul se ocupă cu citirea datelor de intrare, al doilea actualizează pozițiile obiectelor, iar cel de-al treilea se ocupă de afișarea jocului. Astfel, au fost obținute 60 de cadre pe secundă.

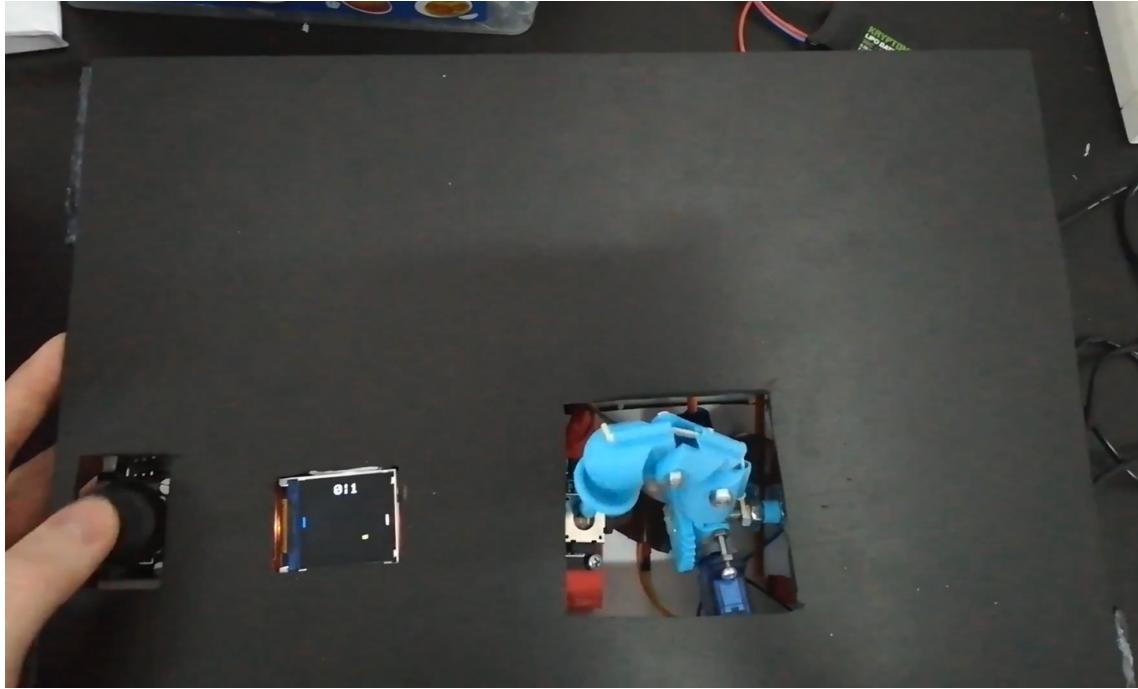
## **II. Brațul robotic autonom**

În acest capitol vom prezenta realizarea robotului intelligent. Acesta este format din cinci subcapitole, iar primul detaliază originile acestui proiect. În al doilea subcapitol vor fi descrise componentele folosite și cum au fost acestea asamblate. În următorul subcapitol vom explica cum au fost preluate și procesate datele observate de camera RGB. Pe baza imaginilor se vor calcula coordonatele paletelor și a mingii.

În cel de-al patrulea subcapitol va fi prezentat algoritmul determinist pentru jucătorul robotic, evoluția acestuia și rezultatele obținute. În ultimul subcapitol vom descrie cum a fost antrenat un agent inteligent de Reinforcement Learning, ce tehnici au fost încercate și rezultatele acestor experimente.

### **II.1 Versiuni anterioare**

Implementarea curentă este o continuare a unui alt proiect. Acesta era unul mult mai simplu, care avea la bază un Arduino Uno. Pe microcontroller era rulat jocul de Pong, afișat pe un ecran și controlat cu ajutorul unor joystick-uri. Tot la Arduino era conectat și un deget robotic, proiectat să controleze al doilea jucător. În această variantă, joystick-ul robotului era puțin modificat pentru ca degetul să nu alunecă de pe el. Toate instrucțiunile robotului erau date cunoșcându-se exact coordonatele obiectelor din joc, iar strategia era una simplă, dar eficientă. Dacă mingea se mișca către adversar paleta trebuia mutată către mijlocul ecranului (pe direcția verticală). În schimb, dacă direcția mingii era către robot, se calcula traiectoria acesteia și punctul exact în care ajungea. Luând în calcul această informație și poziția paletei robotului îi transmiteam acestuia în ce direcție să se miște. Această abordare, reușea să prindă toate mingile. Rezultatul perfect al agentului a condus la încercarea rezolvării unei probleme mai dificile.



*Figura 2.1: Prima versiune a proiectului*

## II.2 Hardware

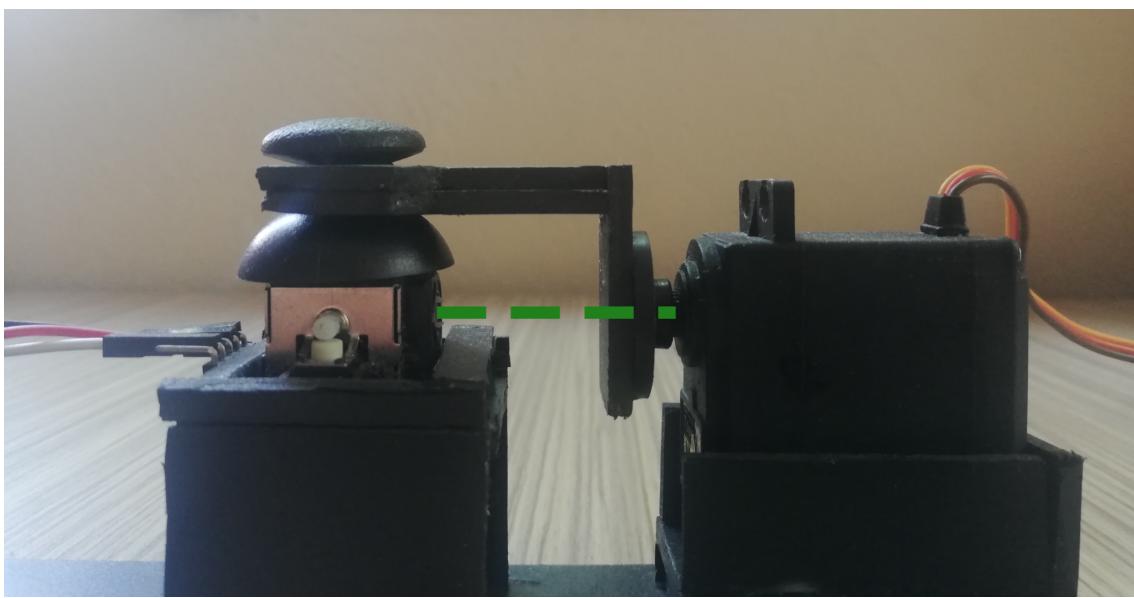
Robotul are drept componentă principală un Raspberry Pi 4B. Datorită dimensiunilor foarte reduse, acesta se poate poziționa foarte ușor unde este nevoie. De asemenea, este destul de puternic din punct de vedere computațional pentru a rula algoritmi de procesare a imaginilor și agenți de Inteligență Artificială.

Mini calculatorul are atașată o cameră RGB. Prin aceasta se vor captura imagini ale jocului pentru a fi procesate și analizate de calculator. Pentru cameră a fost confectionat un suport, astfel încât aceasta să fie îndreptată către ecranul jocului.

Raspberry Pi-ul este legat și la un servomotor, care, la rândul lui are atașat un mini braț robotic realizat din hobbyglas. Brațul nu are părți mobile, iar mișcările acestuia sunt descrise în totalitate de servomotor.



*Figura 2.2: Componentele robotului și suportul*



*Figura 2.3.1: Servomotorul este aliniat cu joystick-ul*



*Figura 2.3.2: Brațul robotic se rotește pe aceeași trajectorie cu joystick-ul*

## II.3 Procesarea datelor

Pentru componenta software a fost folosit limbajul de programare Python 3. Jocul este observat prin intermediul camerei cu ajutorul librăriei *picamera*. Sunt colectate imagini cu rezoluția de  $400 \times 304$  pixeli cu o frecvență de 20 de cadre pe secundă. Calitatea imaginilor a fost redusă cât mai mult, astfel încât numărul de cadre pe secundă să fie cât mai mare, iar procesarea datelor să fie cât mai rapidă. În același timp, dimensiunea lor este suficientă pentru a putea extrage toate datele necesare.

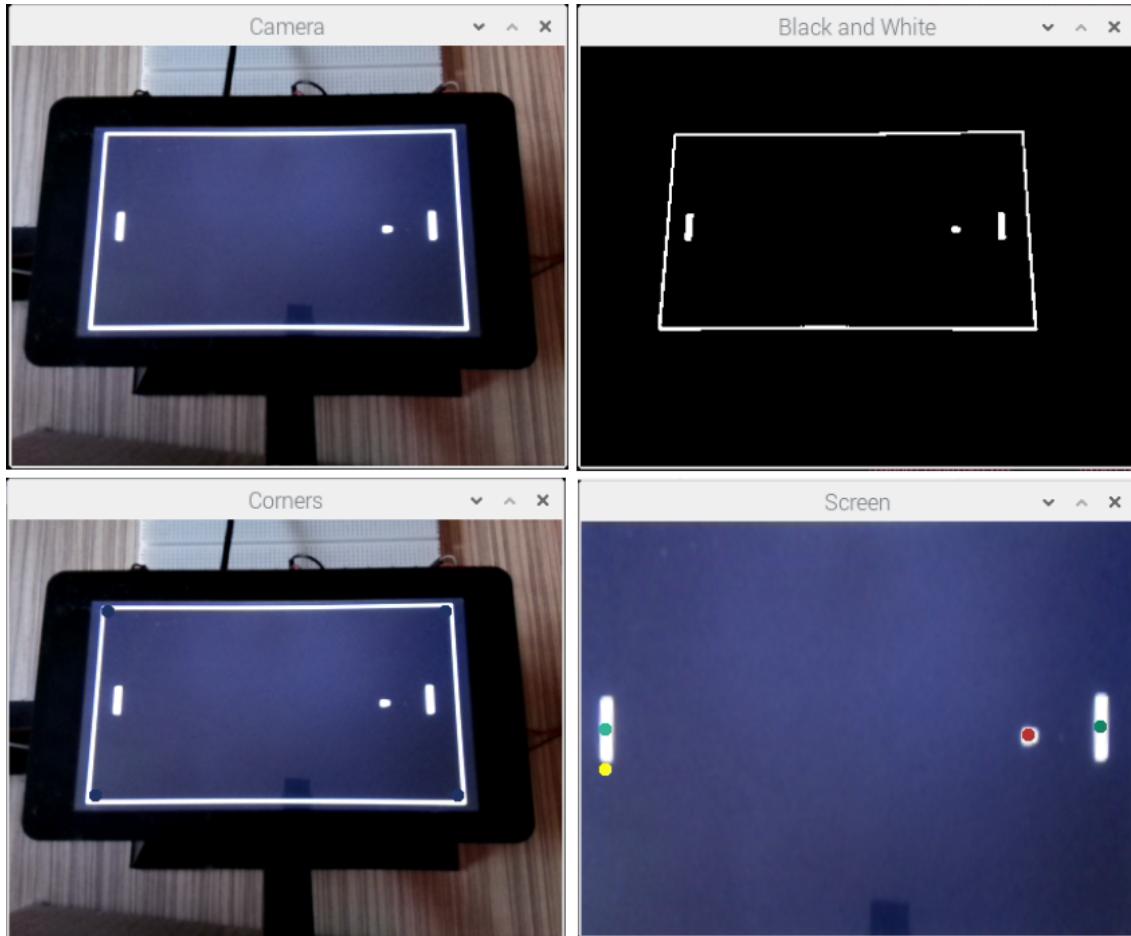
Pentru a scoate în evidență părțile importante ale imaginilor, acestea sunt transformate în format grayscale (tonuri de gri), iar apoi este aplicat un threshold (prag). După acest proces, toți pixelii imaginii au valoarea 1 (alb) sau 0 (negru) și sunt vizibile doar paletele, mingea și marginile jocului.

Pentru primele iterații folosim marginile afișate pentru a determina exact porțiunea în care se petrece jocul. Cu ajutorul bibliotecii *OpenCV* sunt determinate colțurile [5] din imagine și sunt selectate cele patru puncte din extreame. Apoi este creată o matrice de transformare [4] pe baza punctelor anterioare și colțurile unei noi imagini de  $400 \times 300$  pixeli. Aceasta este folosită pentru a transforma [3] zona dintre cele patru puncte selectate într-o imagine nouă cu rezoluția dorită.

Acest proces este realizat doar la începutul rulării programului pentru că este unul costisitor din punct de vedere computațional. Pentru că ne așteptăm ca poziția camerei să nu se miște pe parcursul jocului, vom folosi aceleași date pentru a extrage zona importantă a imaginilor viitoare.

În acest moment avem o imagine alb-negru care conține doar porțiunea jocului. Pentru că am aplicat acești pași nu mai suntem dependenți de poziția inițială a camerei (ne așteptăm să existe mici variații între diferite rulări ale programului).

Pe baza noii imagini, detectăm contururile obiectelor [6] și calculăm centrele lor. Următorii algoritmi se vor baza exclusiv pe aceste coordonate.



*Figura 2.4: Procesarea datelor. De sus în jos, de la stânga la dreapta:  
 imaginea originală, imaginea în alb și negru,  
 colțurile tablei marcate cu puncte albastre, tabla de joc*

În ultima imagine se pot observa câteva puncte colorate. Cele verzi reprezintă centrele paletelor, cel roșu centrul bilei, iar cel galben marchează locul unde credem că va ajunge mingea. Vom detalia calculul traiectoriei mingii în următoarea secțiune.

## II.4 Algoritmul determinist

După procesarea datelor am putut construi mult mai ușor un prim algoritm pentru robot. Luând în considerare ultimele două centre ale bilei am determinat direcția acesteia. Am calculat diferențele pe fiecare axă, iar apoi am împărțit valorile la modulul direcției pe axa orizontală, pentru a normaliza valorile. De exemplu, dacă diferențele sunt  $\Delta x = -5$  și  $\Delta y = 2.5$ , vitezele calculate pentru fiecare axă sunt  $v_x = -1$  și  $v_y = 0.5$ .

Din perspectiva robotului, acesta este în partea stângă. Asta înseamnă că mingea se îndreaptă spre el dacă are viteza orizontală  $v_x = -1$ . În cazul acesta prezicem la ce poziție pe axa verticală va ajunge bila prin calculul  $predicție = (minge_x - paletă_x) \times v_y$ . Acest calcul ignoră, însă, coliziunile cu marginile tablei. Pentru a rezolva problema am simulațat coliziunile cu peretei. De exemplu, predicția inițială 530 ar avea în final valoarea 70 (arena are înălțimea de 300 de pixeli).



### **Figura 2.5: Vizualizarea traectoriei bilei**

Am considerat o eroare  $err$  arbitrară. Dacă predicția aparține intervalului  $[paletă_y - err, paletă_y + err]$  robotul trebuie să rămână pe loc. Dacă predicția este mai sus de  $paletă_y + err$ , sau mai jos de  $paletă_y - err$  instruim robotul să se miște în direcția potrivită.

În cazul în care mingea se îndepărtează de robot, în loc de predicție este folosit mijlocul tablei pe plan vertical. Robotul va muta paleta în centru, fiind pregătit pentru orice lovitură a adversarului. În această versiune mișcam servomotorul 20 de grade la stânga / la dreapta, pentru realizarea mișcărilor.

Această abordare este una bună, însă, robotul ajunge de foarte multe ori să se miște prea mult într-o direcție. Pentru a rectifica greșeala se întoarce, dar, din nou, parcurge o distanță prea mare. Este repetat procedeul până ce mingea ajunge la el, iar rezultatul este unul relativ aleator. Cu acest algoritm a fost observată o rată de succes de 75% împotriva unui oponent uman, unde succesul este definit drept prinderea unei mingi. A fost aleasă această statistică, în locul câștigurilor, pentru că reflectă mai bine scopul programului - adică prinderea cât mai multor bile.

Pentru a îmbunătăți algoritmul au fost modificate trei componente. Prima componentă este mișcarea servomotorului. În momentul actual, brațul este mișcat strictul necesar astfel încât jocul Pong să detecteze mișcarea. Acest lucru este foarte important, pentru că minimizează erorile făcute și îi permite agentului să facă mișcări mult mai rapide. Cum servomotorul nu este perfect calibrat, folosește unghiuri diferite pentru direcțiile dreapta și stânga. În final, unghiurile făcute de acesta sunt congruente și au o măsură de 18 grade.

A doua componentă care a fost modificată este predicția bilei. Pentru a explica algoritmul, pornim de la cele două puncte reținute și un nou punct pe care încă nu l-am folosit. Fie aceste puncte  $a$ ,  $b$  și  $c$ , unde  $a$  este primul punct observat, iar  $c$  este ultimul. De asemenea, fie  $t$  traectoria calculată precedent. Dacă traectoria descrisă de punctele  $b$  și  $c$  este asemănătoare lui  $t$ , înlocuim punctul  $b$  cu  $c$ . În acest moment avem două puncte de pe aceeași traectorie, dar cu o distanță mai mare între ele. Datorită acestei distanțe, traectoria este una mult mai precisă. Dacă noul punct nu aparține aceleiași traectorii, reținem punctele  $b$  și  $c$  și vom calcula traectoria pe baza lor. În acest caz,  $b$  este punctul în care mingea și-a schimbat traectoria,

Prin traectorii asemănătoare se înțelege că direcțiile au același semn, pe ambele axe. Asta înseamnă ca mingea nu a atins alt obiect, pentru că orice coliziune modifică cel puțin

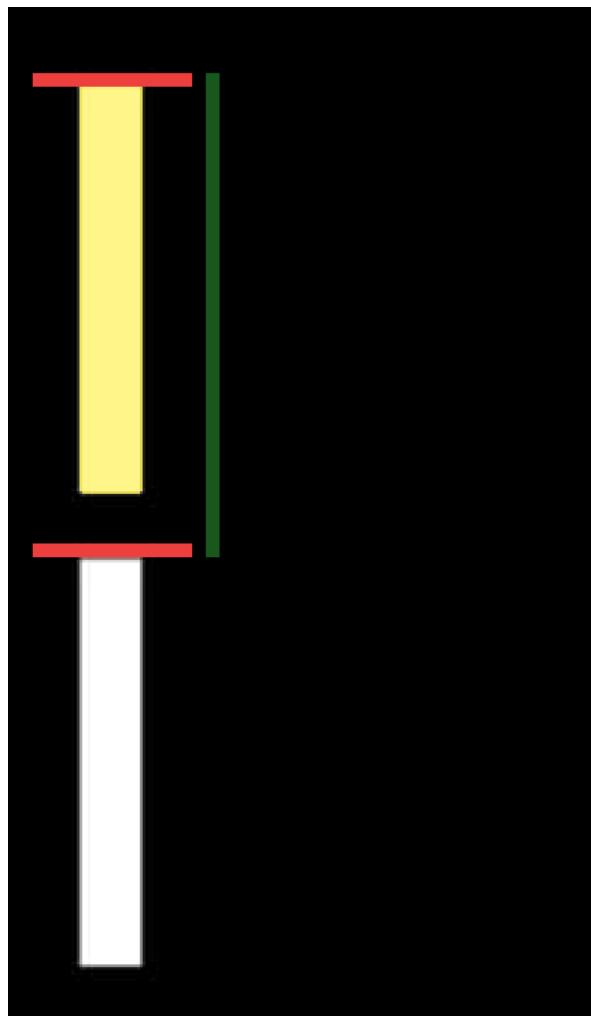
una dintre aceste valori. De exemplu, traiectoriile  $(-1, 0.5)$  și  $(-1, 0.4)$  sunt asemănătoare, dar traiectoriile  $(1, 0.5)$  și  $(1, -0.1)$  nu sunt asemănătoare.

Din acest punct, predicția se calculează la fel ca în versiunea trecută. Folosind această abordare, predicția este semnificativ mai precisă și converge mult mai repede la valoarea reală. În figura 2.5 se poate observa aplicarea acestui algoritm. În imagine există două puncte roșii. Cel din stânga reprezintă poziția actuală a bilei, iar cel din dreapta locul de unde aceasta și-a început traiectoria curentă.

Ultima modificare constă în simularea inerției robotului. Când robotul este înclinat (paleta este mișcată), există o întârziere de când acesta primește instrucțiunea de oprire până când nu mai acționează asupra paletei. Astfel, există o oarecare inerție în mișcarea paletei, iar la momentul actual, aceasta este cauza principală pentru ratarea mingilor. Pentru a rezolva problema, a fost luat în considerare acest efect.

Inițial, considerăm inerția egală cu 0, iar pentru primele cadre în care paleta se deplasează, incrementăm valoarea acesteia cu 1. Apoi înmulțim valoarea cu  $X$  pentru a aproxima distanța, în pixeli, pe care o va parcurge în plus paleta după instrucțiunea “stop”. Valoarea lui  $X$  a fost aleasă în urma experimentelor și este egală cu  $\frac{1}{3}$ . Pentru că această abordare nu este perfectă, am preferat alegerea unei valori mai mici, decât a uneia mai mari. Așa, micșorăm riscul de a-i da agentului date greșite, chiar dacă uneori avantajul produs este unul mai mic.

Apoi adăugăm, sau scădem, în funcție de sens, această valoare la poziția paletei. Rezultatul marchează locul unde credem că se va opri jucătorul. Ilustrăm în figura 2.6 noua logică.



*Figura 2.6: Predicția unei palete care se mișcă în sus*

Bara albă reprezintă paleta jucătorului, iar cea galbenă unde credem că se va opri aceasta. Pentru că jucătorul se mișcă în sus considerăm capetele superioare ale paletelor. Dacă mingea va ajunge între aceste două puncte (în intervalul liniei verzi) îi spunem robotului să se opreasă și lăsăm inerția acțiuneze. Altfel, mutăm brațul robotic mai sus, sau mai jos, după caz.

Cu aceste modificări, robotul reușește să aibă o rată de succes de 91.75% împotriva unui jucător uman, unde rata de succes reprezintă procentajul de bile lovite. Rezultatul agentului actual este mult mai bun decât cel precedent. Acum robotul lovește aproape toate bilele cu viteză mică, sau medie. Bilele ratațe sunt în general din cele cu viteză de deplasare mare (cel puțin 11 din 12). Chiar și la aceste viteze, robotul reușește să prindă multe

mingi decât oponenții săi umani. Chiar dacă agentul nu este perfect, acesta are o performanță foarte bună.

## II.5 Reinforcement Learning

Reinforcement Learning este una din cele trei paradigmă de bază din Machine Learning. Aceasta se ocupă cu modul în care agenții inteligenți ar trebui să acționeze într-un mediu pentru a maximiza o anumită recompensă.

Jocul Pong este unul simplu, iar strategia învingătoare este una clar definită. Din acest motiv a putut fi folosit un algoritm determinist. De multe ori, însă, jocurile au reguli mai complicate, iar strategia optimă nu este una clară. Pentru că ne propunem ca robotul să poată juca o varietate mai mare de jocuri, atât video, cât și fizice am decis să explorăm antrenarea unui agent inteligent. Astfel, instrucțiunile nu mai sunt dictate de noi, ci alese de model pe baza experienței. În acest fel, robotul nu mai este dependent de abilitatea noastră de a găsi și implementa o strategie.

Programul a fost scris în limbajul Python 3. În primul rând, am construit un mediu de lucru care simulează jocul de Pong cu ajutorul librăriei *gym* [2] de la OpenAI. Acesta se comportă exact ca jocul descris mai sus. Jucătorul 1 (din stânga) funcționează după un algoritm simplu, și anume urmărirea bilei. El încearcă mereu să mențină paleta la nivelul acesteia. Acest algoritm a fost ales pentru că este unul decent, însă care poate fi învins de un jucător bun. Mingea se poate deplasa mai repede decât paleta, iar, în acest caz, jucătorul nu poate ține pasul.

Pentru realizarea mediului, cele mai importante metode care au trebuit implementate au fost *reset* și *step*. În funcția *reset* jocul este adus la starea de început, iar în funcția *step* este iterat un pas al jocului. Cea din urmă primește drept argument direcția aleasă de agent (0 - mutare în sus, 1 - stat pe loc, sau 2 - mutare în jos). Pe baza acesteia se efectuează un pas al programului (fiecare obiect este mutat pe baza direcției și vitezei lui) și sunt returnate starea jocului, recompensa și dacă episodul s-a terminat.

### II.5.1 Deep Q-Learning

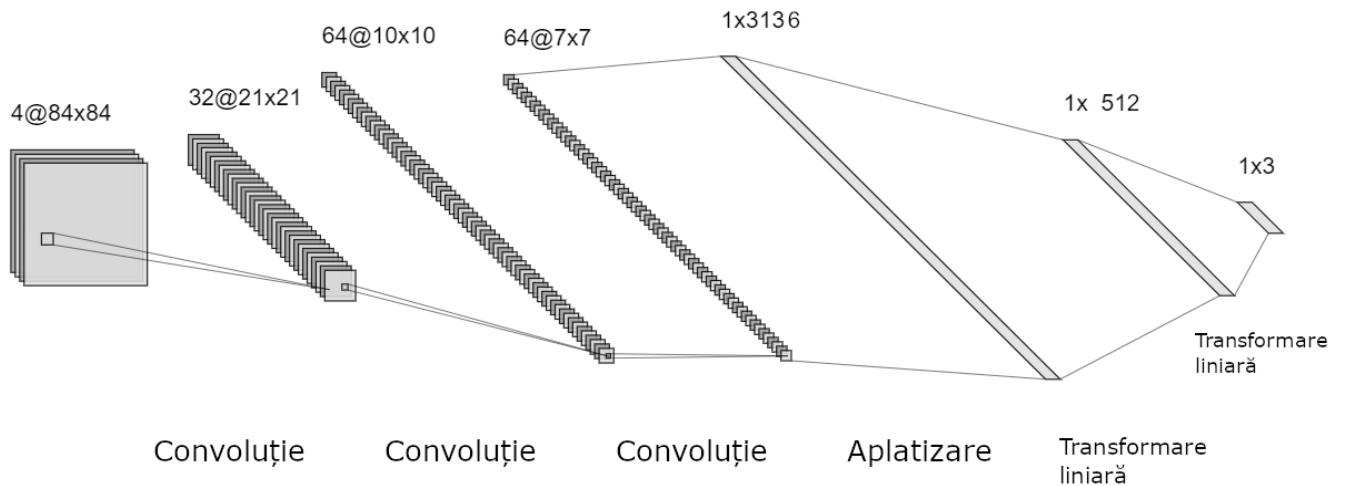
Vom explica algoritmul Q-Learning într-un mod simplificat. Pornim de la premisa că avem o funcție *R* care primește două argumente (*S* și *A*), primul reprezentând o stare a jocului, iar al doilea o acțiune. Funcția întoarce recompensa imediată pe care se așteaptă să o

primească dacă în starea  $S$  ia acțiunea  $A$ . De exemplu: dacă prin acțiunea  $A$  se ajunge din starea  $S$  în starea finală  $S'$  care oferă recompensa 10, atribuim lui  $R(S, A)$  valoarea 10. De asemenea, considerăm funcția  $Q(S, A)$  care întoarce recompensa pe care ne așteptăm să o primim până la finalul jocului. În cazul precedent,  $Q(S, A)$  are tot valoarea 10. Pentru a deosebi cele două funcții luăm în considerare starea  $S''$ , prin care se poate ajunge la starea  $S'$  prin acțiunea  $A'$ , primind recompensa 5.  $R(S, A)$  ar returna valoarea 5, iar  $Q(S, A)$  ar avea valoarea 15. În practică, recompensele imediate au mai multă importanță decât cele prezise în viitor. Adică, în exemplul precedent,  $Q(S, A)$  ar avea o valoare cuprinsă între 5 și 15, în funcție de factorul de reducere. Funcția  $Q$  ar fi inițializată cu 0 și s-ar actualiza după următoarea ecuație.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R_{t+1} + \gamma Q(S_{t+1}, A') - Q(S_t, A_t)),$$

unde  $\gamma$  reprezintă factorul de reducere,  $\alpha$  este rata de învățare, iar  $t$  reprezintă tură curentă. Pentru a afla acțiunea optimă la momentul curent sunt verificate valorile  $Q(S_t, A_t)$  pentru toate acțiunile posibile  $A_t$  și este aleasă acțiunea care rezultă în cel mai bun rezultat. La început, acțiunile sunt alese aleator, iar pe măsură ce agentul explorează, este aleasă tot mai des acțiunea optimă.

Pentru rezolvarea problemei am folosit algoritmul Q-Learning cu o rețea neuronală conlovuțională. Prima abordare a constat în antrenarea agentului pe baza imaginilor, iar episoadele se terminau în momentul în care un jucător marca un punct. Starea jocului constă în reprezentarea alb-negru a arenei (o imagine cu un singur canal). Aceste imagini au fost redimensionate la  $84 \times 84$  de pixeli, iar valorile normalizate (cea mai mare valoare fiind 1). Așa, imaginile pot fi procesate mult mai rapid, dar încă includ toată informația scenei. Pentru a folosi algoritmul de Q-Learning, procesul trebuie să fie unul Markovian. Astfel, toate informațiile despre joc trebuie să fie descrise într-o singură observație. Aceasta nu este cazul pentru folosirea unei singure imagini pentru că nu se pot determina direcțiile obiectelor. Din acest motiv, agentul primea ultimele 4 imagini observate. Modelul constă într-o rețea neuronală conlovuțională, având următoarea arhitectură:



**Figura 2.7: Arhitectura rețelei neuronale convolutionale**

*Figură realizată cu ajutorul <http://alexlenail.me/NN-SVG/LeNet.html>*

După fiecare operație de convecție și de transformare liniară a fost aplicată funcția de activare ReLU pe toate valorile, unde  $ReLU(x) = (x)^+ = \max(0, x)$ . Ieșirea rețelei este o listă cu trei elemente și reprezintă “încrederea” cu care ar face acțiunea respectivă. Cele 3 acțiuni sunt mișcare în sus, stat pe loc și mișcare în jos. De exemplu, rezultatul [0.1, 0.4, 0.5] înseamnă că agentul preferă acțiunea 2 (mutare în jos). Suma elementelor din această listă va fi mereu egală cu 1. Pentru a determina acțiunea, se ia poziția elementului cu valoarea maximă. Agentul a fost lăsat la antrenat pentru aproximativ 8 ore, timp după care agentul nu mai progresă.

## II.5.2 Rezultate și Experimente

Agentul inteligent descris anterior avea performanțe bune în mediul virtual. În majoritatea cazurilor reușea să învingă strategia banală a jucătorului 1 și avea o rată de succes decentă contra unui oponent uman. Însă, când programul a fost utilizat drept “creierul” robotului, acesta a avut rezultate foarte slabe. În majoritatea scenariilor, robotul altera constant între direcțiile sus și jos, iar uneori se ducea și după mingă.

Am încercat penalizarea agentului când acesta făcea schimbări brusă ale direcției (mișcare în sus, urmărită imediat de mișcare în jos, sau invers), însă nu a avut un impact asupra performanței.

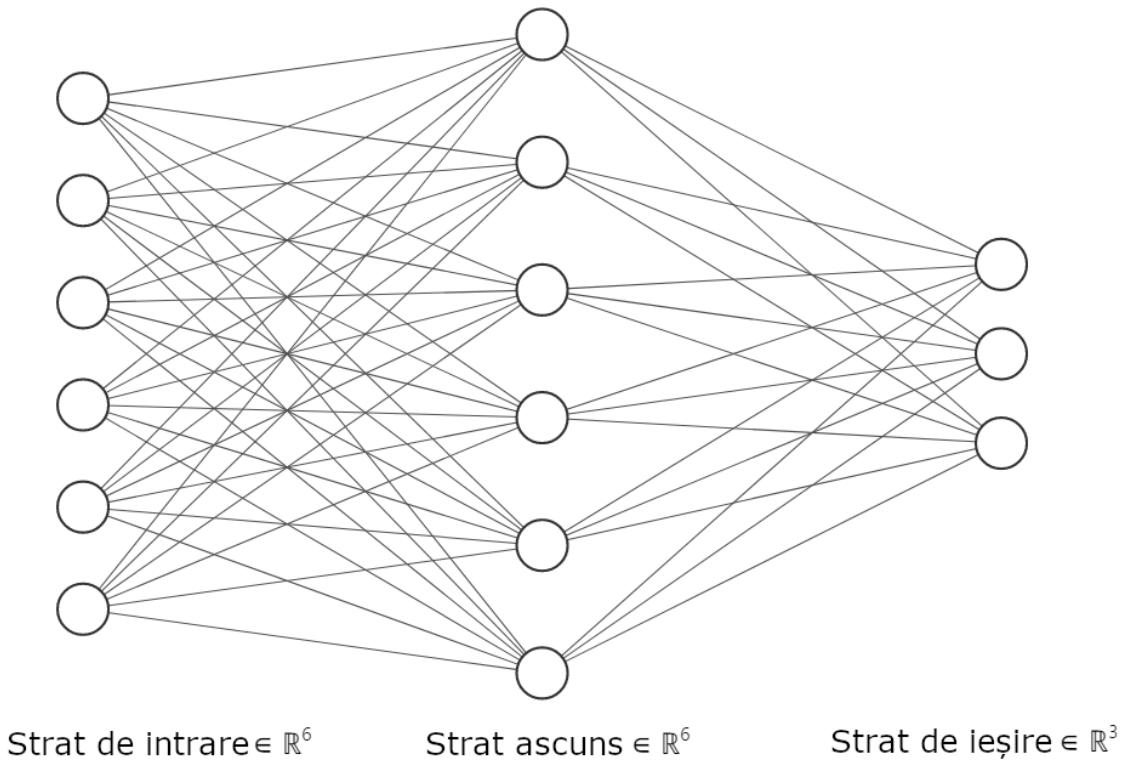
Agentul a fost antrenat apoi în alt mediu, în care episoadele se terminau la fiecare lovitură, sau ratare a robotului. Această modificare a fost realizată pentru a scurta durata de învățare a agentului. În această configurație, acesta era încurajat să joace foarte defensiv, singurul lucru relevant fiind dacă lovește sau nu mingea. De asemenea, pentru a nu limita experiența robotului la câteva scenarii, poziția și direcția de start ale bilei erau aleator la fiecare rulare. Prima variantă avea recompense predefinite pentru prinderea sau scăparea mingii, dar în această variantă, penalizarea pentru ratarea bilei era direct proporțională cu distanța acesteia față de paletă. Astfel, în loc să se deplaseze aleator până ce nimereala mingea, agentul a putut realiza mult mai repede că este un lucru bun să se apropie de aceasta. Timpul de învățare a scăzut considerabil, iar performanța a fost una mai bună decât cea precedentă. În același timp, rata de succes a robotului în mediul fizic a rămas tot mică.

Credem că aceste rezultate slabe se datorează diferențelor dintre cele două medii (cel virtual și cel fizic). Chiar dacă agentul primește imaginile procesate ale arenei, obiectele au dimensiuni și forme diferite. În mediul virtual, acestea sunt pătrate sau dreptunghiulare, având colțurile bine definite. În mediul fizic, din cauza limitărilor camerei, obiectele par mai rotunde, cu margini neclare, iar mingea, la viteze mari, lasă o urmă în spatele ei. Agentul nu înțelege aceste date, iar rezultatele rețelei sunt de cele mai multe ori aleatoare.

### **II.5.3 Folosirea coordonatelor**

Modelul a fost modificat, astfel încât să folosească coordonatele obiectelor drept date de intrare. Acestea constă în coordonatele bilei, coordonatele paletelor robotului și direcția bilei (separată pe cele două axe). Din experimentul cu robotul determinist s-a dovedit că algoritmul de extragere a acestor date este unul de încredere. Datele rezultate din procesarea imaginilor (coordonatele obiectelor) sunt foarte apropiate de cele reale. Așadar, agentul ar trebui să învețe mult mai repede și mai bine pentru că facem o parte din treaba lui. Acum agentul primește doar 6 valori în loc de 4 imagini cu dimensiunea de  $84 \times 84$  de pixeli și, într-adevăr, agentul a avut un progres mult mai rapid de învățare.

Modelul a fost modificat astfel încât să folosească noile date, iar arhitectura acestuia este una mult mai simplă comparativ cu cea precedentă.



*Figura 2.8: Arhitectura noii rețele*

*Figură realizată cu ajutorul <http://alexlenail.me/NN-SVG/index.html>*

Datorită extragerii manuale a datelor, diferențele dintre medii sunt mult mai mici acum. Acum, rezultatele robotului în mediul fizic sunt mai bune. În același timp, performanța acestuia nu este la fel de bună ca a agentului determinist. Acest lucru este de așteptat, având în vedere că în mediul de antrenare nu este luată în considerare întârzierea joystick-ului.

O altă diferență dintre agentul antrenat și cel determinist este stilul de joc. Noul robot apare să fie mult mai uman. Acesta nu mai urmărește un tipar, iar mișcările sunt mult mai spontane. Agentul are o performanță de 51.68% contra unui oponent uman.

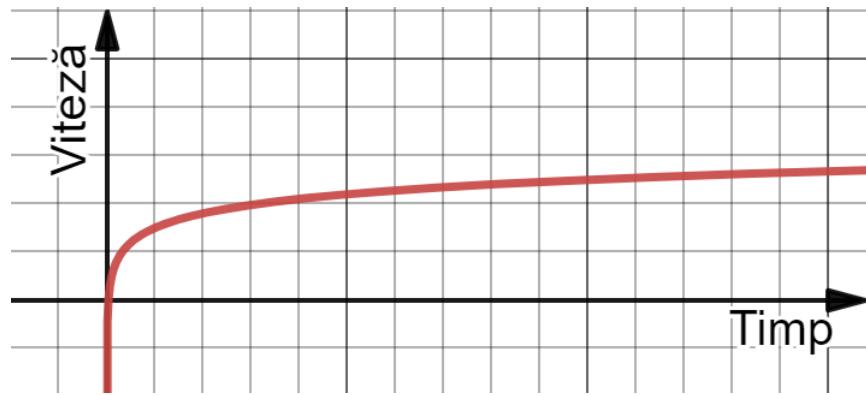
### **III. Rezultate și Concluzii**

În următorul tabel se pot observa performanțele agenților. Numerotarea cu 1 reprezintă prima variantă descrisă a agentului, iar numerotarea cu 2 reprezintă varianta finală a agentului respectiv. Astfel, se poate observa care a fost impactul schimbărilor realizate pe parcursul proiectului. Rata de succes reprezintă procentajul de bile prinse de către agent. De exemplu, dacă robotul a lovit 3 bile consecutive și a ratat-o pe a patra, acesta are o rată de succes de 75%.

	Agentul Determinist 1	Agentul Determinist 2	Agentul Inteligent 1	Agentul Inteligent 2
Rata de succes	75%	91.75%	31.18%	51.68%
Numărul de observații	68	97	61	89

În lucrare se observă că cele mai bune performanțe au fost obținute de robotul determinist. Acest rezultat s-a datorat cunoașterii în amănunt a jocului și a ajustărilor făcute manual. Chiar dacă agentul inteligent are performanțe mai slabe, acesta are un comportament mult mai uman și natural. De asemenea, acest agent ar putea fi îmbunătățit în continuare.

Primul lucru pe care îl propunem să îl modificăm este simularea componentelor fizice în mediul de antrenare. Acest lucru s-ar putea realiza prin înregistrarea timpilor necesari pentru diferitele mișcări ale servomotorului și introducerea lor în mediu. Aceste mișcări ar fi: cele din poziția de stop în pozițiile sus / jos, și invers; cea din poziția sus în poziția jos, și invers. Toate aceste măsurători sunt necesare pentru că servomotorul are și el o anumită inerție. Astfel, începutul mișcării este lent, iar viteza cu care acesta se rotește crește pe parcurs până la o anumită valoare. În trecut am simulat acest comportament al servomotorului cu o funcție logaritmică, însă rezultatul era doar o aproximare a valorii reale. Din acest motiv este de preferat folosirea timpilor în loc de simularea efectivă a dispozitivului.



*Figura 3.1: Aproximarea vitezei de rotație a servomotorului*

# Bibliografie

- [1] Adafruit. “ADS1x15 Library.” [https://github.com/adafruit/Adafruit\\_Python\\_AM2301](https://github.com/adafruit/Adafruit_Python_AM2301). Accessed 2020.
- [2] OpenAI. “OpenAI Gym.” <https://gym.openai.com/>. Accessed 2021.
- [3] OpenCV. “Apply Perspective Transformation.”  
[https://docs.opencv.org/master/d4/d54/group\\_\\_imgproc\\_\\_transform.html#gaf73673ae8e18ec6963e3774e6a94b87](https://docs.opencv.org/master/d4/d54/group__imgproc__transform.html#gaf73673ae8e18ec6963e3774e6a94b87). Accessed 2020.
- [4] OpenCV. “Calculate Perspective Transform.”  
[https://docs.opencv.org/master/d4/d54/group\\_\\_imgproc\\_\\_transform.html#ga20f62aa3235d869c9956436c870893ae](https://docs.opencv.org/master/d4/d54/group__imgproc__transform.html#ga20f62aa3235d869c9956436c870893ae). Accessed 2020.
- [5] OpenCV. “Corner Detection.”  
[https://docs.opencv.org/master/dd/d1a/group\\_\\_imgproc\\_\\_feature.html#ga1d6bb77486c8f92d79c8793ad995d541](https://docs.opencv.org/master/dd/d1a/group__imgproc__feature.html#ga1d6bb77486c8f92d79c8793ad995d541). Accessed 2020.
- [6] OpenCV. “Find Contours.”  
[https://docs.opencv.org/3.4/d3/dc0/group\\_\\_imgproc\\_\\_shape.html#ga17ed9f5d79ae97bd4c7cf18403e1689a](https://docs.opencv.org/3.4/d3/dc0/group__imgproc__shape.html#ga17ed9f5d79ae97bd4c7cf18403e1689a). Accessed 2020.
- [7] PyGame. “PyGame.” <https://github.com/pygame pygame>. Accessed 2020.
- [8] Texas Instruments. “Analog-to-Digital Converter.”  
<https://cdn-shop.adafruit.com/datasheets/ads1015.pdf>.

## Listă figuri

Figura 1.1: Jocul pong.....	8
Figura 1.2: Sistemul pentru Pong.....	9
Figura 1.3: Carcasa jocului.....	10
Figura 1.4: Datele care descriu obiectele.....	11
Figura 1.5: Jocul Pong.....	12
Figura 2.1: Prima versiune a proiectului.....	15
Figura 2.2: Componentele robotului și suportul.....	16
Figura 2.3.1: Servomotorul este aliniat cu joystick-ul.....	16
Figura 2.3.2: Brațul robotic se rotește pe aceeași trajectorie cu joystick-ul.	17
Figura 2.4: Procesarea datelor.....	19
Figura 2.5: Vizualizarea trajectoriei bilei.....	20
Figura 2.6: Predicția unei palete care se mișcă în sus.....	22
Figura 2.7: Arhitectura rețelei neuronale convoluționale.....	25
Figura 2.8: Arhitectura noii rețele.....	27
Figura 3.1: Aproximarea vitezei de rotație a servomotorului.....	29