https://doi.org/10.37815/rte.v33n2.852
Artículos originales

Artificial intelligence-controlled pole balancing using an Arduino board

Balanceo de un poste, controlado por una inteligencia artificial usando una placa Arduino

José Revelo[1] https://orcid.org/0000-0003-1159-1523, Oscar Chang[1] https://orcid.org/0000-0003-1241-1782

[1]*Yachay*, Imbabura, Ecuador
jose.revelo@yachaytech.edu.ec, oscar.chang@yachaytech.edu.ec

Sent: 2021/07/11
Accepted: 2021/09/28
Published: 2021/11/30

Resumen

El proceso de automatización (AP) actual es de gran importancia en el mundo digitalizado, en rasgos generales, representa un aumento en la calidad de producción con el trabajo hecho a mano. El equilibrio es una capacidad natural del ser humano que está relacionada en trabajos y conducta inteligente. Equilibrarse representa un desafío adicional en los procesos de automatización, debido a la presencia de múltiples variables involucradas. Este artículo presenta el equilibrio físico y dinámico de un poste en el que un agente, mediante el uso de aprendizaje por refuerzo (RL), tiene la capacidad de explorar su entorno, detectar su posición a través de sensores, aprendiendo por sí mismo cómo mantener un poste equilibrado bajo perturbaciones en el mundo real. El agente usa los principios de RL para explorar y aprender nuevas posiciones y correcciones que conducen a recompensas más significativas en términos de equilibrio del poste. Mediante el uso de una matriz Q, el agente explora las condiciones futuras y adquiere información de política que hace posible mantener el equilibrio. Todo el proceso de entrenamiento y pruebas se realizan y gestionan íntegramente en un microcontrolador Arduino. Con la ayuda de sensores, servo motores, comunicaciones inalámbricas e inteligencia artificial, todos estos componentes se fusionan en un sistema que recupera constantemente el equilibrio bajo cambios aleatorios de posición. Los resultados obtenidos demuestran que a través de RL un agente puede aprender por sí mismo a utilizar sensores, actuadores genéricos y resolver problemas de balanceo incluso bajo las limitaciones que presenta un microcontrolador.

Palabras clave: aprendizaje por refuerzo, agente inteligente, microcontrolador, Industria 4.0.

Sumario: Introduction, Related work, Methodology, Results and Conclusions.

Como citar: Revelo, J. & Chang, O. (2021). Artificial intelligence-controlled pole balancing using an Arduino board. *Revista Tecnológica - Espol, 33(2), 189-204.*
http://www.rte.espol.edu.ec/index.php/tecnologica/article/view/852

Abstract

Automation Process (AP) is an important issue in the current digitized world and, in general, represents an increase in the quality of productivity when compared with manual control. Balance is a natural human capacity as it relates to complex operations and intelligence. Balance Control presents an extra challenge in automation processes, due to the many variables that may be involved. This work presents a physical balancing pole where a Reinforcement Learning (RL) agent can explore the environment, sense its position through accelerometers, and wirelessly communicate and eventually learns by itself how to keep the pole balanced under noise disturbance. The agent uses RL principles to explore and learn new positions and corrections that lead toward more significant rewards in terms of pole equilibrium. By using a Q-matrix, the agent explores future conditions and acquires policy information that makes it possible to maintain stability. An Arduino microcontroller processes all training and testing. With the help of sensors, servo motors, wireless communications, and artificial intelligence, components merge into a system that consistently recovers equilibrium under random position changes. The obtained results prove that through RL, an agent can learn by itself to use generic sensors, actuators and solve balancing problems even under the limitations that a microcontroller presents.

Keywords: reinforcement learning, intelligent agent, Q-learning, microcontroller, Industry 4.0.

Introduction

Factories continue to improve and introduce new technology to reduce their production costs, such as the solution given to repetitive work in the delivery of goods industry (Azadeh et al., 2019). New technologies have to target and solve two main aspects to keep progressing and improve tasks in multiple areas: transparency of costs that makes replication easier and produce these technologies; recreate a more human-like robot behavior.

One primary aspect and basic act that robots should replicate from humans, is to be proficient at balancing. Whether we want a humanoid or a computer to replace the workers, they have to be capable of performing on a similar level to humans (Hyon et al., 2007). Ships' displacement is often dependent on careful balancing; in sea navigation, equilibrium is critical. The balancing mechanism is based on fin stabilizers, which rotate and change the fin angle to resist ocean disturbance (Sun et al., 2018). Balance is also present in aviation. Unmanned aerial vehicles (UAVs) use stabilizers that play an important role minimizing disruption effects and ensuring a smoother flight (Korkmaz et al., 2013), by dealing with three different angles known as yaw, roll, and pitch. Balancing is a critical component that affects the ability to perform a wide variety of tasks and is essential in the operation require by loaders, robotic arms, ships, anti-seismic systems, or robots themselves.

Machine activity does not present adaptation or reaction to new events that occur, and are not predictable during normal function. This is critical if correct balancing is a goal in changing environments. Unpredictable events and varying conditions of the environment are why automation has limited uses in many factories and moving vehicles. AI may be set to perform a task in varying circumstances; it may give improved results, and be efficient in activities currently executed only by humans. A contemporary and relevant tool in AI is Reinforcement Learning (RL), a methodology by which agents independently learn efficient control policies. They then use this knowledge to solve logical or mechanical problems (Foerster et al., 2017).

The first reason to use Arduino is its flexibility to adapt to different projects. AI requires data recollection and this may be a difficult task because data is proprietary or cost prohibitive. In this sense, Arduino is handy and very helpful for recollecting data from the world.

Arduino is reasonably priced for building several prototypes. Also, as shown in (López-Rodríguez & Cuesta, 2016; Taha & Marhoon, 2018), is how feasible it is to build more than one specific model or prototype with customizable devices, depending on the situation. The low cost advantage also decreases barriers to perform a study in different areas of AI devices, or reducing problems to replicate and produce multiple similar technologies that instantly interact with the real world.

With these premises, this paper develops a dynamic balancing pole that AI manages running in an Arduino microcontroller. Here, an RL agent, with the capacity to read sensors and control the angles of servo motors that drive an x, y balancing pole, learns by itself how to keep the pole balanced under noise disturbance, responding to natural changes of its environment. The prototype demonstrates the capabilities of microcontrollers that have some limitations, compared with regular desk computers, and how extensive is there potential.

Related work

The following section examines and evaluates works that are relevant to the planted objectives and share common themes, strategies, and method elaboration.

Arduino fully managed systems

Due to their low and medium cost, Arduino-based projects can be produced or recreated. Due to easy access to a variety of hardware resources, it is simple to execute multi-purpose projects. A mobile robotic platform is presented in (Araújo et al., 2015). In the work, it is claimed that robots must be as inexpensive as possible for students and researchers to conduct real-world experiments. The work in (Ram et al., 2017) demonstrates the versatility of Arduino when used in an IoT project, a proper power supply, wireless communication and sensor devices are good features of working with Arduino boards. The Arduino microcontroller is used in (González & Calderón, 2019) to build a Supervisory Control and Data Acquisition (SCADA) system with high configuration possibilities, easy access to libraries and a fast learning rate for new users. In (Wu et al., 2017) it is described the Arduino board's data collection, tracking, and scalability projection capabilities.

The majority of the work discuss the low-cost advantages of Arduino projects, as well as how to adapt or include more electronic devices to create more robust devices with no issues. Furthermore, the majority of the works incorporate two elements: automation and real-time operations. Arduino can conduct real-time operations without the use of simulations, concentrating on the problems that are important to the target, enabling it to detect planning mistakes or missed steps in the early stages of development. This has been a big help for researches, both technically and theoretically, showing that Arduino is a very useful tool.

Artificial intelligence with microcontrollers

In (Jain, 2018), a self-driving car is demonstrated using a Raspberry Pi and a Convolutional Neural Network. An Arduino board guides the movements of the vehicle. It receives an order and drives the car in a specific direction. The author concludes that the car's design and testing were successful. In addition, the work (López-Rodríguez & Cuesta, 2016) shows a mobile robot for educational purposes based on Android and Arduino. The robot has a light sensor, GPS, camera, accelerometer, Bluetooth, and Wi-Fi, and can accept commands from a smartphone via an Internet connection. The author claims that software alteration can

be easily applied for future projects. In the third study (Lengare & Rane, 2015), image processing is used to monitor the movements of a human arm and reproduce this action in a robotic arm controlled by an Arduino. The author claims that an Arduino board can monitor the robot's behavior; however, there are many ways to accomplish the same task.

It is clear from the previous work that the Arduino will perform actions with good results if the instructions come from a logical process of an efficient AI or a human. The argument is that with the use of a microcontroller, the majority of the work present no hardware problems. As a consequence, with the right instructions, Arduino will perform a variety of tasks. The intelligence of data analysis is executed outside the microcontroller in all previous projects, but as further projects illustrate, it is also possible to code the intelligence or incorporate learning techniques in the Arduino itself, without having to raise costs or resources.

Artificial Intelligence has shown better results than conventional approaches in several of projects; these works involve intelligent agents that perform several tasks in pursuit of optimum results. It was shown in (Jimenez et al., 2020; Salazar et al., 2013), that an intelligent agent's capable of considering timing, amount of water, and properly implementing them in what they detail as Spatio-temporal variations of the soil–plant–atmosphere system, gathering impressive results in terms of water efficiency and precision irrigation. As agents work with microcontrollers like Arduino, they may create further applications. For example, (Mata-Rivera et al., 2019) shows an intelligent traffic light control device that uses Arduino sensors to detect the color of traffic lights, as well as the tone, distance, and motion of vehicles in the streets where it is installed. The entire process is executed in Arduino, demonstrating how this platform can perform complex tasks using machine learning techniques.

Since the recently mentioned works used an Arduino to carry out the whole operation, these project reinforces the concept that Arduino hardware and software are plenty capable, and assisting this study in realizing that AI can be implemented in a microcontroller. Of course, Arduino has limitations, but the previous works only require a little extra effort to comprehend the obstacles, adapt the prototype, and include RL.

Reinforcement learning in Real environments

The aim of developing robots is to allow them to participate in a real-world environment where unexpected events can occur. The goal is to make machines more effective in performing various tasks in a manner that is equivalent to, if not superior to, a human. To accomplish this action in computers, a variety of techniques are used.

Three classifications of methods are given in (Kormushev et al., 2013), as well as mentioned how computers can learn to deal with such tasks. Direct programming is defined as the lowest method. This is more akin to a programming method than it is to learning. The second method is imitation learning, which is more comparable to a learning strategy. The problem with this approach is that it requires a specialist to perform a perfect presentation of how to complete the job. The last method, RL is defined as a trial-and-error method that explores the environment and the robot's body. RL has three benefits over the other two approaches: learning new tasks, optimizing efficiency, and adapting to new situations.

RL has been used in real-world settings to solve physical problems in various experiments, but simulations can also be used in training. According to (Pan et al., 2017), depending on the context, the training will present undesirable driving behaviors that can cause harm to the environment. As a result, depending on the case, a simulation or real-world environment may be used to train a computer. The problem with transitioning from a virtual to

a real-world environment is that it takes extra time to apply the learned skills to a different scenario than the one in which they were educated. The research of (Miglino et al., 1995) demonstrates how simulations can solve problems that appear to be closer to the original target but are not; in other words, the simulation environment lacks realistic behavior. However, conducting training in the real world poses a challenge; as mentioned in (Nagabandi et al., 2018), samples can be extremely costly. The tools used can influence this complication about cost of collecting samples. As mentioned, an Arduino's main advantage is its low cost.

The work from (Gu et al., 2017) creates and trains a robot to open a door from scratch, which is a realistic application that requires complex contact dynamics to imitate human physical abilities. In addition, it demonstrates how real robotic platforms can perform physical tasks in 3D real environments. It is also claimed that RL is an effective training method for these systems. The work of (Sharma et al., 2020) uses a free-reward RL algorithm to teach a robot how to travel properly within its structure and navigate, 20 hours of preparation, the machine had mastered a variety of locomotion gaits. After a short time, the RL methods yielded results. This research showed how RL can be used to teach an AI how to control its movements in the real world, without wasting too much time in the training process.

Methodology

This section describes the proposed system, which learns to detect a properly stable position. The construction of the balancing prototype has three main points to be described, divided into the following sections: General description of the system. A detailed description of the transmitter device. A detailed description of the receiver device. In the end, generating an algorithmic description of the training process.

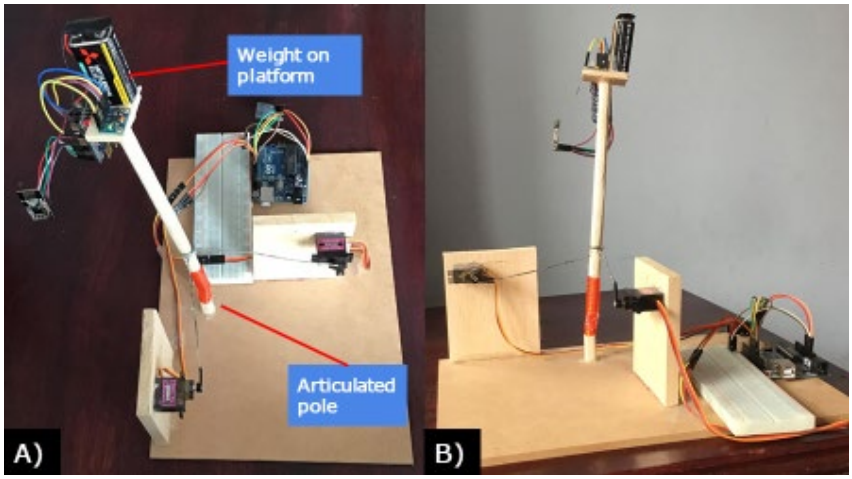
Problem Solving Phases

Problem Description

The structure to balance consists of a pole with a platform at the top, where objects can be placed. The pole cannot stand up straight due to a flexible component in the base, the pole is connected to this component which allows the pole to constantly bend in any direction. To control the pole displacement, two servo motors are connected using a rigid wire. One servo motor moves the pole up and down (north-south), the second moves the pole from left to right (east-west). An upper view of the balancing structure shown in Figure 1 A, and a front view in Figure 1 B.

Figure 1

A View of the Structure to Balance



Analysis of the Problem

The first issue is to translate the environment in terms that is possible to convert into a programming platform, select the correct data type, and attempt not to exceed the memory capacity of the Arduino.

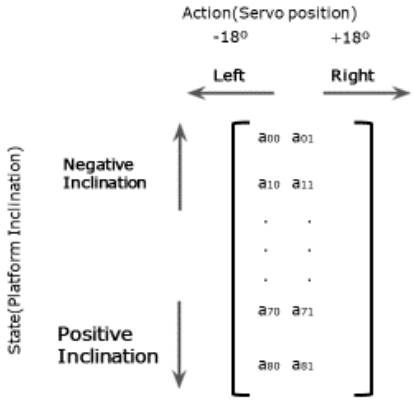
The speed execution of the code is enough to sustain the speed movement of the servo motors. To train an agent and measure a reward, is necessary to measure after a change or move is completed. The Arduino board can execute multiple moves virtually before a servo motor actually has finished the first move. In other words, the measurement has some imprecision issues, due to movements that are supposed to happen but are retarded because of inertial forces.

Implementation

The agent's work is to randomly explore the environment through its own actions. The agent will try different moves depending of its actual state, meaning that the agent will increase and decrease the angle in the servo, allowing the agent to explore which action will yield better or worse results, depending on the detected state. The Q-matrix, also known as a Q-table, represent the conditions and possible behavior of the agent represented. The dimensions of this Q-matrix are 9x2, represented in Figure 2, where the number of columns represents the number of possible actions from which the pole will pass. The number of files represents the inclination states in which the platform will be.

Figure 2

Q-Matrix, States(Rows) and Actions (Columns)

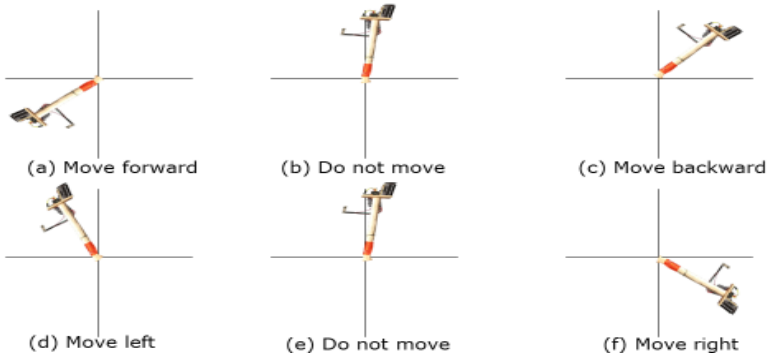


The servo motors in use operate in degrees from 0º-180; one pushes the pole forward or backward Figure 3 (a), while the other moves the pole left Figure 3 (c), or right Figure 3 (d). However, if the system detects that is in balance, both servo motors will not move pole Figure 3 (b) and Figure 3 (e). The first column depicts the action of decreasing the current angle at which the servo motor is located, while the second column depicts the action of raising the current angle at which the servo motor is located. The servo motor has 144º of inclination since the eighth row is 8, which is the highest value that can be found in this matrix beginning at state 0 from 0º, in the servo motor.

The error to balance the pole is 18º due to the Q-matrix design. Both servos increase or decrease its angle 18º, the angle was chosen to save space memory in Arduino, and also the servo motors have a better response with angle variations of a larger magnitude, one-degree variation will be a good improvement, but better precision servo motors are required.

Figure 3

Possible actions performed by the servos



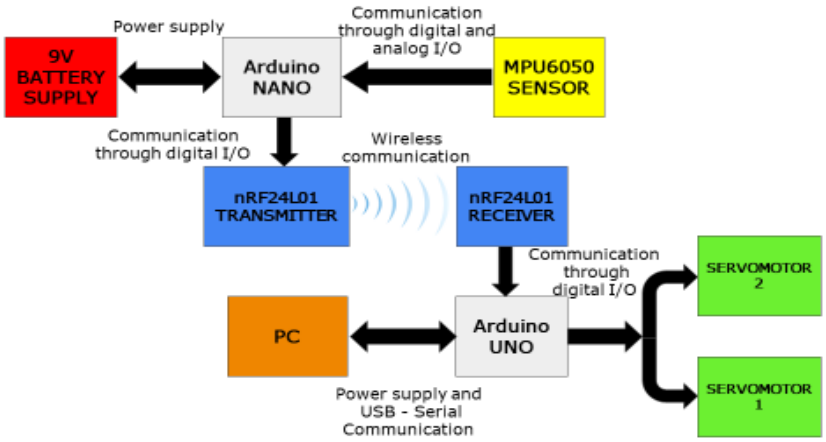
The prototype stabilizes the post in 18 steps, with each servo performing nine steps simultaneously, in a worst-case scenario. Delays measured in the servo's movement speed, allowing the balancing time to increase or decrease. Servos perform angle variations every 200 milliseconds; hence these nine moves can be completed in less than 1800 milliseconds. In the worst-case scenario, the prototype's balancing time is nearly two seconds.

System description

The prototype design is structured by two devices working together to balance the pole, as seen in Figure 4. This details which component is connected and how it communicates and flows. It is known as a Q-table, from upper components to which are below. the device A and B have wireless communication, the implementation and a deeper description of the internal process of each device will be described in the following sections. Device A, due to its structure, could be reused in more areas to estimate more precise position changes. The device B does not have the same reuse capacity; it has to be modified depending on the balanced structure.

Figure 4

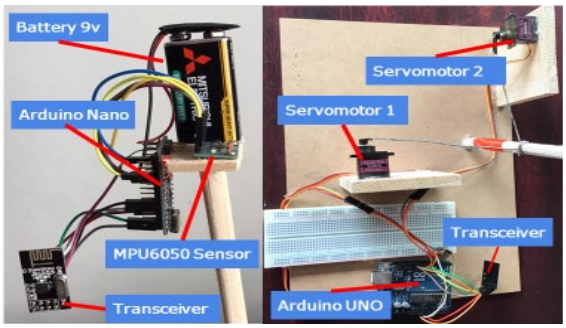
Prototype Whole Process Flow Diagram



Device A

Has the task to recollect and notify position changes of the platform in which it is placed. Figure 5 shows the electronic elements used as the Arduino Nano, the accelerometer and gyroscope that are in the MPU6050 sensor, and the nRF24L01 RF transceiver. Device A performs the flow process in Figure 6.

Figure 5

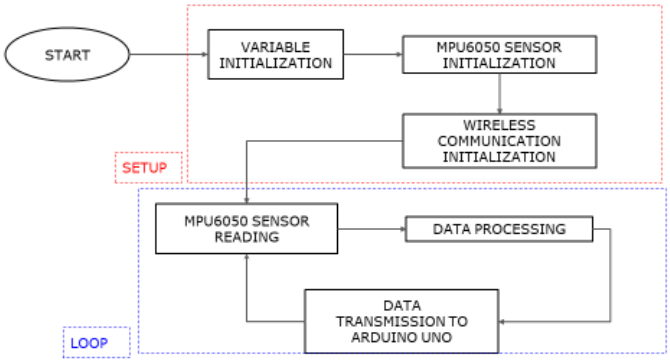


Device B

This device, managed by Arduino Uno, will execute the process from Figure 7. Afterwards, the Train process finishes and will execute the steps from Figure 8. Device A in Figure 5 shows the two servo motors, that manage the positions of the pole, also have the nRF24L01 RF transceiver for wireless communication.

Figure 6

Arduino NANO Process Flow Diagram



Device A process

Setup and Loop Functions

The setup and loop functions are the two main functions of most Arduino programs. After uploading the program to an Arduino microcontroller, the setup function is the first to run. The loop function is a loop that repeats the actions inside the microcontroller until the microcontroller resets or the power supply is disconnected. Both functions are required in Arduino programs.

Sense and Transmission Flow Process

The Arduino Nano carries out the entire process depicted in Figure 6. First, is the setup function, where all variables of the accelerometer and gyroscope in the three-axis x, y, and z are initialized in the variable initialization step. The data rate in symbols per second (baud) has several bits per second (bps) in which the Arduino will communicate during the transmission of data is setup. Sensor initialization begins the MPU6050 functionality after ensuring that it is operational. The Power Amplifier level, the channel between 2.400 and 2.524 GHz, the data rate, and the address of the receiver to which we will send data, are all set up in the wireless communication initialization step.

The loop, as seen in Figure 6, is the second part that constantly reads position changes. MPU6050 is constantly read during sensor reading, and the data obtained is then sent to data processing, where all information is processed in degrees and g (terms of standard gravity). After that, data transmission begins, which reports the changes in the pole's extreme to the Arduino UNO.

Device B process

Train Flow Process

Setup in Figure 7 and Figure 6 are very similar, except for the variables initialization step with the R and Q matrices for the training process. As seen in Figure 6, use the same parameters for wireless communication initialization.

In the loop section, the data reception is emitted from the accelerometer in the Arduino Nano. Action Selection consists of a random selection of the available options from the servo motors. Now Perform action executes the selected action from the previous step. Calculating a reward is the next step, updating the Q-table. A fixed number of iterations is achieved all the steps are repeated. The end section is saving the experience that is in the Q-table, earned in the loop section for forthcoming testing.

Figure 7

Arduino UNO Train Phase Process Flow Diagram

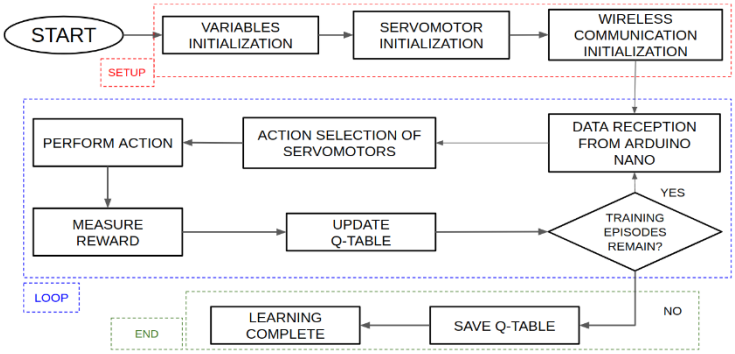
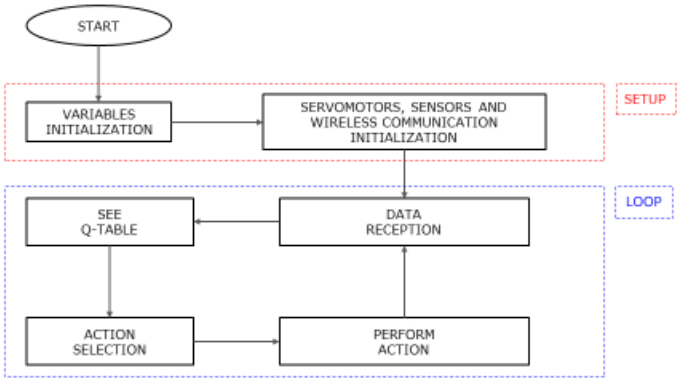


Figure 8



Testing Flow Process

Setup in Figure 7 and Figure 6 show the same steps. Then in the loop function, the data reception is received from the Arduino Nano, then it checks the Q-table, selects the best option,

and performs the action selected in the servo motors establishing the balance and repeating the process from the data reception.

Code structure

The pseudocode from Figure 9 is train agent and executed to train the agent. The input of the algorithm are the variables that describe the current state of the environment.

First are initialized the different variables the *Qm* is Q-matrix and all the values are initialized from zero, which will present changes in the values while learns the policy, *K* is the number of limited episodes, *Moves* is the variable that limits the number of moves an agent can execute before finish an episode.

After the train starts, first are selected an *x* and *y* position, a random number selected between the 0 and the number of columns for *x* and the number of files for *y* of the *Qm* matrix, the variable *Agent* is assigned a random starting action from *Qm*[*x*][*y*] position.

The following inner loop works as a limited sequence of steps where the agent can move around the environment until it reaches the goal or executes a limited number of moves defined previously in the variable *Moves*. The function StepSelection looks at matrix *Qm* neighborhood values, selects an action of the possible options. The possible options of the act allow the agent to move in four possible ways; up, down, left, right.

Updating the variables *Agent, x, y,* depend on the previous move. The variable State saves the new environment variables. The matrix Qm updates the index *x*, *y* value calling the function reward, which uses the variables *State* and *action* to calculate the new value function Reward.

If the agent reaches the goal state the episode finishes, otherwise this inner loop is repeated until the end of the allowed number of steps. The output is a Q-table matrix, the *Qm* which has the experience obtained after the agent exploration.

Figure 9

Agent Training Pseudocode

Algorithm 1: Train agent

Input: "State" = Environment variables

Initialize of the Q matrix Qm, Initialize Qm; Initialize number episodes K;
Initialize number of moves $Moves$;
for $i \leftarrow 0$ ***to*** K **do**
    x=Random() and y=Random();
    $Agent = Qm[x][y]$;
    for $j \leftarrow 0$ ***to*** $Moves$ **do**
        take an action a,;
        $(step_x, step_y)$ = StepSelection(action);
        $Agent = Qm[x + step_x][y + step_y]$;
        $State'$= Current environment variables;
        $Qm[x][y]$= Reward $(action, state)$;
        if $CurrentStateBalanced() == True$ **then**
            break;
        end
    end
end

Output: Q-table matrix

Results

Table 1 shows the Q-table values for the servo motor that pushes the pole left and right, while Table 2 shows the Q-table values for the second servo motor's forward and backward. The table depicts the Q-table during various training episodes, and it is possible to observe how the absolute value rises. The 0 value does not change during training because the episode ends when it enters this state; this state is considered the most stable because it receives a reward of 0, while the other states receive a punishment depending on the platform's inclination. It is easy to see how values that are farther from the stable state are punished more severely, resulting in extreme values gaining more knowledge, or learning faster than states closer to the optimal state. As compared to Q-tables with more iterations, it is possible to see that certain values in one column that should be lower than the other column are not. In the first 50 episodes in the table Q-table from Table 1, it was determined that the agent does not have the experience to choose the proper actions at this point in the training.

Table 1

Agent 1 Q-table Evolution of Values at Different Training Episodes

EPISODE	100		500		1000	
State	Actions					
0	-7.63	-7.63	-16.42	-16.42	-16.13	-16.13
1	-7.59	-5.66	-17.45	-13.32	-17.44	-13.32
2	-3.91	-3.06	-12.95	-9.45	-12.96	-8.38
3	-1.44	-1.76	-6.56	-4.94	-8.48	-3.93
4	0	0	0	0	0	0
5	-0.86	-0.54	-1.00	-2.33	-1.00	-2.75
6	-3.10	-4.99	-3.90	-7.04	-3.90	-6.77
7	-7.04	-9.41	-8.51	-13.99	-8.51	-12.37
8	-9.49	-9.49	-12.66	-12.66	-12.66	-12.66

Table 2

Agent 2 Q-table Evolution of Values at Different Training Episodes

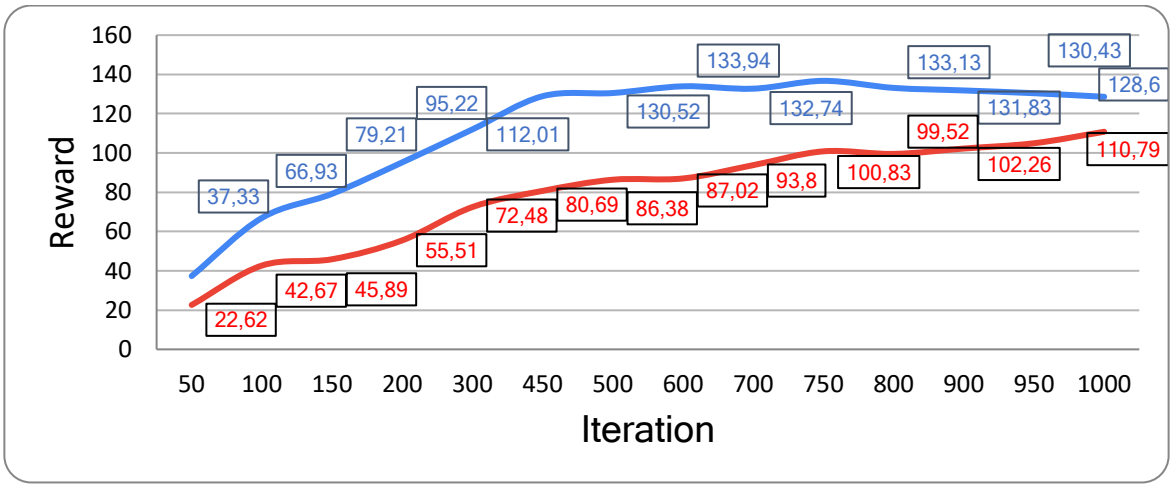
EPISODE	50		100		500		1000	
State	Actions							
0	-3.254	-3.54	-3.254	-3.54	-9.02	-9.02	-14.30	-14.30
1	-2.15	-1.73	-17.45	-1.73	-10.21	-8.22	-15.15	-13.49
2	-3.35	-2.24	-12.95	-2.24	-9.13	-7.74	-13.52	-10.74
3	-3.75	-2.77	-6.56	-2.77	-9.04	-3.99	-11.75	-6.32
4	0	0	0	0	0	0	0	0
5	-0.83	-0.47	-0.83	-0.47	-1.00	-1.46	-1.00	-1.67
6	-1.18	-1.73	-1.18	-1.73	-2.68	-3.38	-1.90	-3.40
7	-2.68	-4.69	-2.68	-4.69	-3.71	-7.44	-3.71	-6.50
8	-5.78	-5.78	-5.78	-5.78	-7.34	-7.34	-7.34	-7.34

In the cumulative reward Figure 10, the x-axis represents the number of episodes trained, while the y-values represent the absolute value total of each variable corresponding to the Q-table. The graphic demonstrates that the agent is learning, gaining experience after each iteration from the first to the nth episode. Figure 10 plots a summary of all the learning work,

both curves present a general growth doing decently. Over the last two hundred iterations agent 1 does not have a noticeable increase in comparison to earlier episodes. Agent 2 shows that it could improve its experience, this behavior comes from the random aspect of the training process, at some episodes would have a noticeable increase in comparison to earlier episodes, this behavior comes from the random aspect of the training process, at some episodes will have a noticeable.

Figure 10

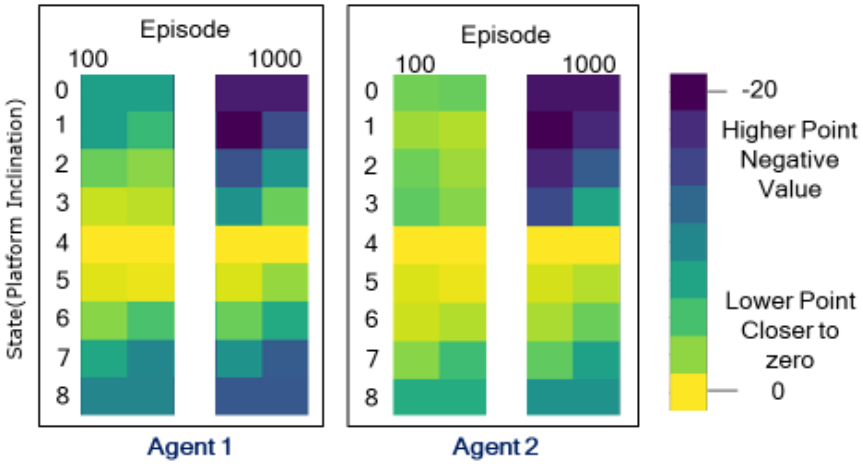
Cumulative Reward as a Function of the Number of Episodes, from both Agents



In Figure 11 are the Q-matrix at episode 100 and 1000, of both agents. The colors with a more negative value represent a higher point and have a darker color, while values closer to zero are more clear and lower points. The agent's path will be the clearest option where it would be located, at episode 1000, the disparity in colors between the different states of the Q-matrix is more apparent compared at episode 100, where the path to follow is somewhat diffuse.

Figure 11

Agent 1 and Agent 2 Heatmap Q-matrix representation



The pole is at the leftmost position; Figure 12 A depicts the direction in which the servo will move the pole; internally, the agent determines which alternative is most appropriate. Figure 12 B is a basic example that concludes when the pole is balanced.

Figure 12

Prototype balancing the pole simple case

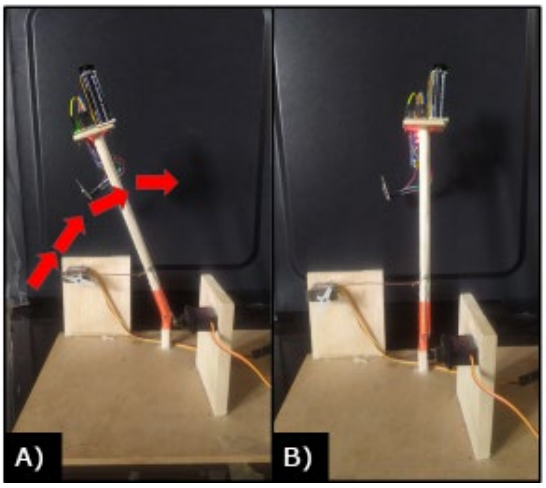


Figure 13 shows an interesting behavior, which an external disturbance is manually introduced, tilting the base, similar to what happens in air and sea ships. The agents must now solve a fresh and unexpected problem.

In Figure 13 A, B, C, the agent 2, tilted back, is in charge of balancing the pole in a forward-backward motion. Agent 2 begins to increase the servo's angle by 18 degrees until it is balanced in this direction; nevertheless, this movement has influenced the pole's correct position in the left-right direction; as a result, agent 1 begins to tilt the pole to the left, Figure 13 D, E, and F depict the prototype's last steps in a real-world scenario.

Figure 13

Prototype behavior tilting the platform

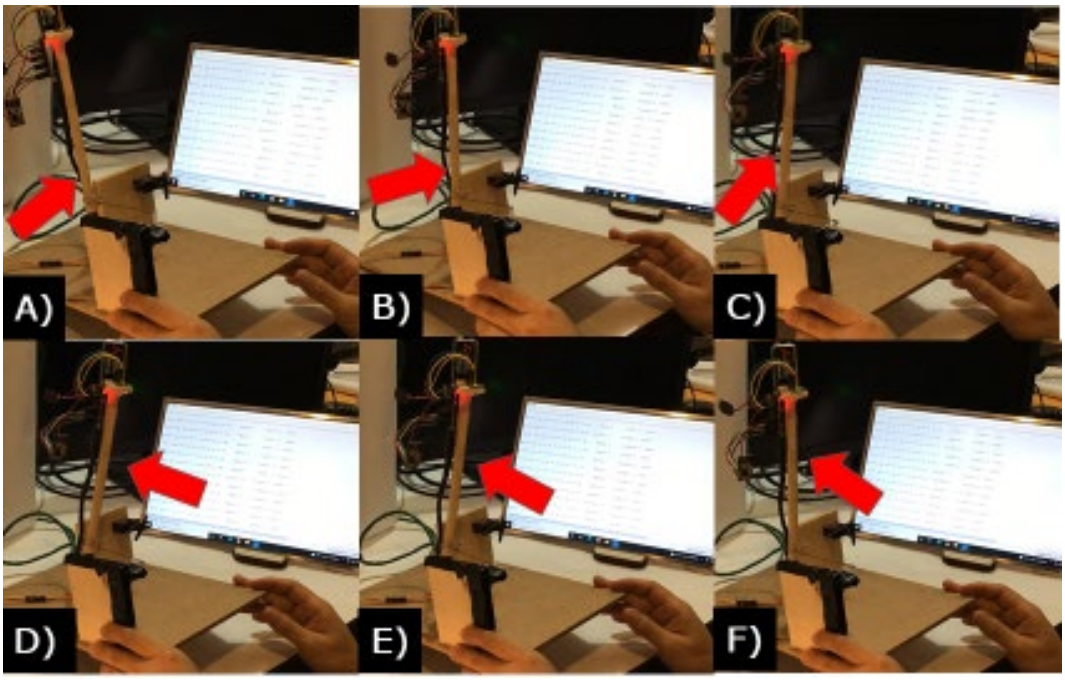
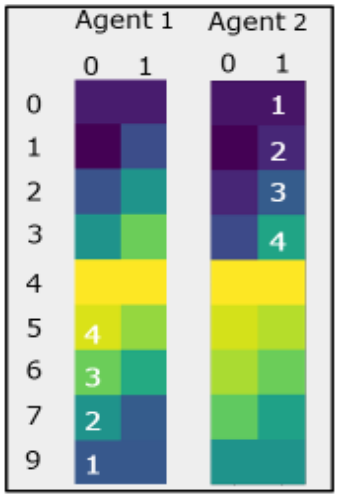


Figure 14 is an internal representation of the actual behavior after finishing the training following a path and choosing the clearest option, depicting the direction that agent 1 will take if it is in the rightmost state, a right inclination. Agent 1 will gradually shift the pole to the left until it reaches the state in row 4. The agent 2 detects that is at a state 0, thanks to the sensometer, which means the platform is facing backward, and it will shift to the front by increasing the signal sent to the servo motor by 18º, moving from state 0 to state 4.

Figure 14

Internal Behavior of Agents



Conclusions

This project includes a study of Arduino microcontrollers, some of their peripherals, and the implementation of a RL agent in a completely Arduino-based prototype consisting of two devices that operate by wireless communication. Within the experimental limitations, the automation process implementation with Artificial Intelligence and RL agents running in Arduino boards, presented very promising results that, in principle, can extend to more complex intelligent machines in the real world. This will require increasing the number of variables, the resolution of the sensors, the power of the actuators, etc. According to the obtained results during the project's execution, the following facts can be concluded:

- The restricted memory space of the Arduino for storing variables is not a barrier to running learning RL algorithms. In particular, Q-learning algorithms can be fully implemented in current Arduino microcontrollers, as demonstrated.
- In the real world, most environments are analog; however, as demonstrated in this study, using Q-learning and a finite Q-matrix it is possible to convert analog changes in discrete signals and use them to control a pole balancing situation.
- Wireless communication adds value to Arduino projects by allowing them to collect data and/or send instructions in complex environments with limited space and/or constant movement. Also, opens possible valuable applications in the world of the Internet of things (IoT).
- Agents and RL algorithms are powerful tools because they allow robots to train themselves in many tasks, such as maintaining dynamic balance using only their implemented components such as sensors and actuators.

- Arduino platforms have access to a variety of peripherals with methodical inclusion, and is a fertile ground to work and develop self-learning macro sensors, robots, and control systems.

Referencias

Araújo, A., Portugal, D., Couceiro, M. S., & Rocha, R. P. (2015). Integrating Arduino-based educational mobile robots in ROS. *Journal of Intelligent \& Robotic Systems*, *77*(2), 281–298.

Azadeh, K., De Koster, R., & Roy, D. (2019). Robotized and automated warehouse systems: Review and recent developments. *Transportation Science*, *53*(4), 917–945.

Foerster, J., Nardelli, N., Farquhar, G., Afouras, T., Torr, P. H. S., Kohli, P., & Whiteson, S. (2017). Stabilising experience replay for deep multi-agent reinforcement learning. *International Conference on Machine Learning*, 1146–1155.

Garcia, J., & Shafie, D. (2020). Teaching a humanoid robot to walk faster through Safe Reinforcement Learning. *Engineering Applications of Artificial Intelligence*, *88*, 103360.

González, I., & Calderón, A. J. (2019). Integration of open source hardware Arduino platform in automation systems applied to Smart Grids/Micro-Grids. *Sustainable Energy Technologies and Assessments*, *36*, 100557.

Gu, S., Holly, E., Lillicrap, T., & Levine, S. (2017). Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates. *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 3389–3396.

Hyon, S.-H., Hale, J. G., & Cheng, G. (2007). Full-body compliant human--humanoid interaction: balancing in the presence of unknown external forces. *IEEE Transactions on Robotics*, *23*(5), 884–898.

Jain, A. K. (2018). Working model of self-driving car using convolutional neural network, Raspberry Pi and Arduino. *2018 Second International Conference on Electronics, Communication and Aerospace Technology (ICECA)*, 1630–1635.

Jimenez, A.-F., Cardenas, P.-F., Canales, A., Jimenez, F., & Portacio, A. (2020). A survey on intelligent agents and multi-agents for irrigation scheduling. *Computers and Electronics in Agriculture*, 105474.

Korkmaz, H., Ertin, O. B., Kasnako\uglu, C., & others. (2013). Design of a flight stabilizer system for a small fixed wing unmanned aerial vehicle using system identification. *IFAC Proceedings Volumes*, *46*(25), 145–149.

Kormushev, P., Calinon, S., & Caldwell, D. G. (2013). Reinforcement learning in robotics: Applications and real-world challenges. *Robotics*, *2*(3), 122–148.

Lengare, P. S., & Rane, M. E. (2015). Human hand tracking using MATLAB to control Arduino based robotic arm. *2015 International Conference on Pervasive Computing (ICPC)*, 1–4.

López-Rodríguez, F. M., & Cuesta, F. (2016). Andruino-A1: Low-Cost Educational Mobile Robot Based on Android and Arduino. *Journal of Intelligent and Robotic Systems: Theory and Applications*, *81*(1), 63–76. https://doi.org/10.1007/s10846-015-0227-x

Mata-Rivera, M. F., Zagal-Flores, R., & Barría-Huidobro, C. (2019). *Telematics and Computing: 8th International Congress, WITCOM 2019, Merida, Mexico, November 4--8, 2019, Proceedings* (Vol. 1053). Springer Nature.

Miglino, O., Lund, H. H., & Nolfi, S. (1995). Evolving mobile robots in simulated and real environments. *Artificial Life*, *2*(4), 417–434.

Nagabandi, A., Clavera, I., Liu, S., Fearing, R. S., Abbeel, P., Levine, S., & Finn, C. (2018). Learning to adapt in dynamic, real-world environments through meta-reinforcement learning. *ArXiv Preprint*

ArXiv:1803.11347.

Pan, X., You, Y., Wang, Z., & Lu, C. (2017). Virtual to real reinforcement learning for autonomous driving. *ArXiv Preprint ArXiv:1704.03952*.

Ram, S. A., Siddarth, N., Manjula, N., Rogan, K., & Srinivasan, K. (2017). Real-time automation system using Arduino. *2017 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*, 1–5.

Salazar, R., Rangel, J. C., Pinzón, C., & Rodríguez, A. (2013). *Irrigation system through intelligent agents implemented with arduino technology*.

Sharma, A., Ahn, M., Levine, S., Kumar, V., Hausman, K., & Gu, S. (2020). Emergent real-world robotic skills via unsupervised off-policy reinforcement learning. *ArXiv Preprint ArXiv:2004.12974*.

Sun, M., Luan, T., & Liang, L. (2018). RBF neural network compensation-based adaptive control for lift-feedback system of ship fin stabilizers to improve anti-rolling effect. *Ocean Engineering*, *163*, 307–321.

Taha, I. A., & Marhoon, H. M. (2018). Implementation of controlled robot for fire detection and extinguish to closed areas based on Arduino. *Telkomnika*, *16*(2), 654–664.

Wu, D., Liu, S., Zhang, L., Terpenny, J., Gao, R. X., Kurfess, T., & Guzzo, J. A. (2017). A fog computing-based framework for process monitoring and prognosis in cyber-manufacturing. *Journal of Manufacturing Systems*, *43*, 25–34.