

# Construcción y desarrollo de un Robot Móvil de Seguimiento, controlado mediante Aprendizaje por Refuerzo Profundo y Visión Artificial

Patrichs Alexander Inocente Valle<sup>1</sup>

<sup>1</sup>Universidad Nacional de Ingeniería. Facultad de Ciencias.  
Ciencia de la Computación

December, 2023



**FACULTAD DE  
CIENCIAS**

# Introducción

- Un robot autónomo de seguimiento tiene la tarea de identificar objetivos a los cuales persigue con la finalidad de alcanzarlos.
- Las áreas de aplicación principales son: La seguridad y vigilancia, asistencia personal a ancianos y asistencia en tareas humanas que requieran una mano de obra demandante como lo son las tareas agrícolas.
- Los robots de seguimiento para mantenerse en diferentes terrenos y escenarios deben absorber la mayor cantidad de información de su entorno mediante sus sensores, y además dotarlo de algunos mecanismos autónomos para poder realizar ciertas tareas, como la detección de obstáculos y enfoque de objetivos mediante visión artificial.



Figura: Robot BURRO AI. Robot de seguimiento para actividades Agrícolas.

# Objetivo

El principal objetivo del presente trabajo de investigación se enfoca en construir y entrenar un Robot móvil físico de configuración diferencial en la tarea de seguimiento de Objetivos, con el debido entrenamiento virtual que le proporcione un correcto movimiento para evitar colisionar con elementos del entorno que puedan poner en peligro al robot.

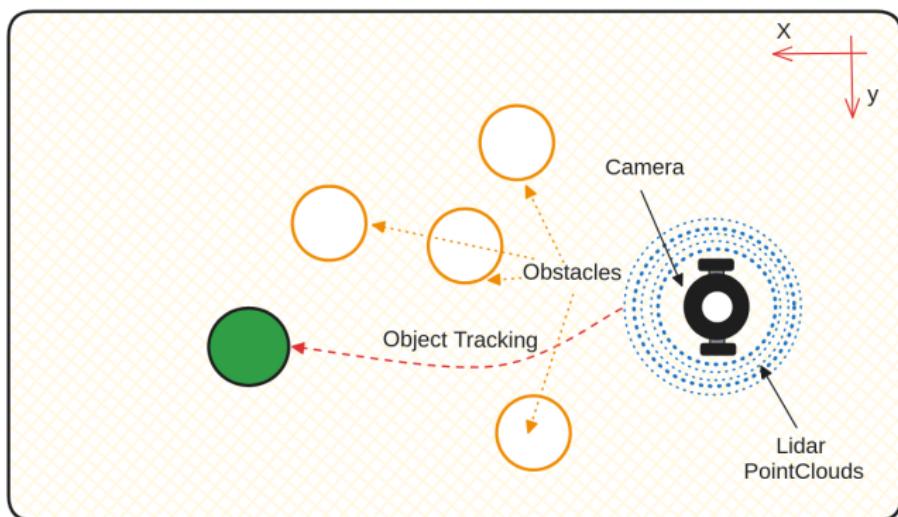
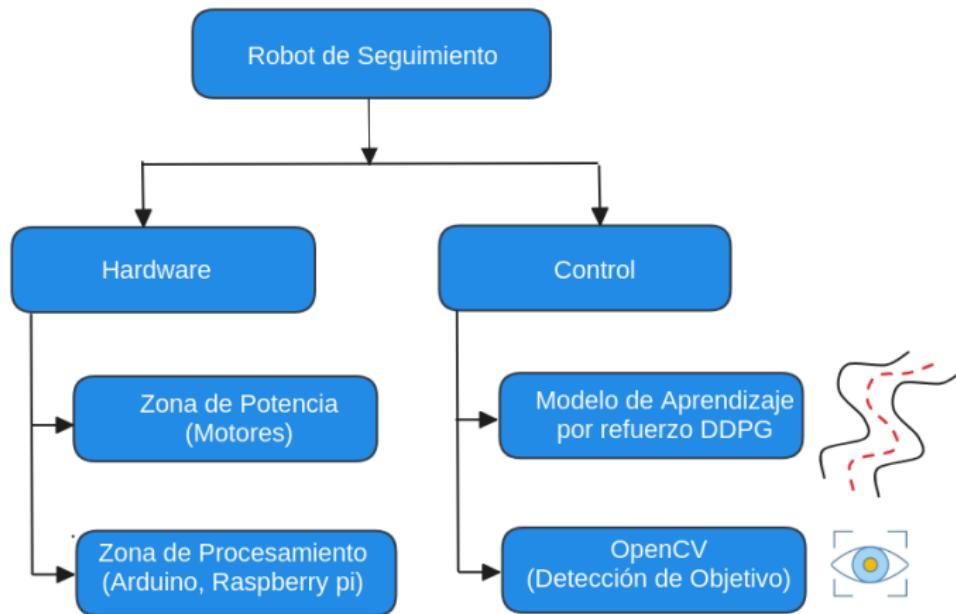
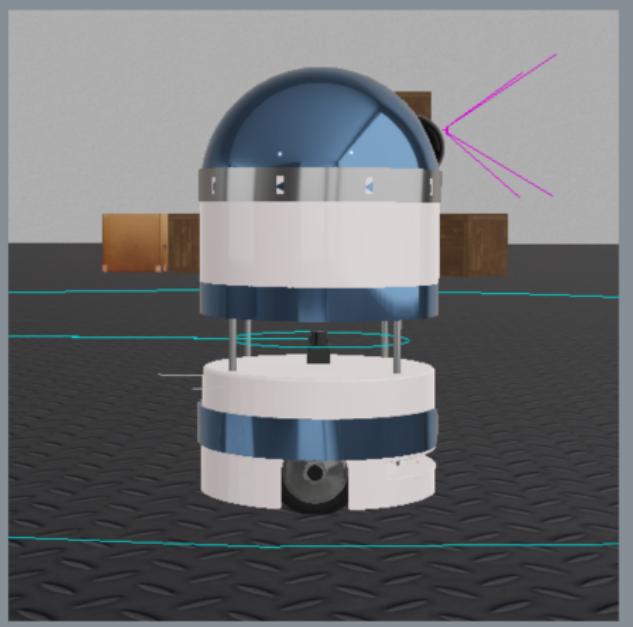


Figura: Planteamiento de Objetivo

# Roadmap del Presente Seminario

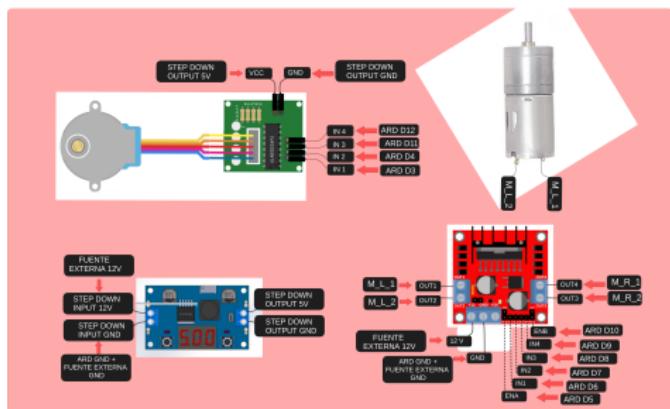


# Hardware [Modelo Virtual]

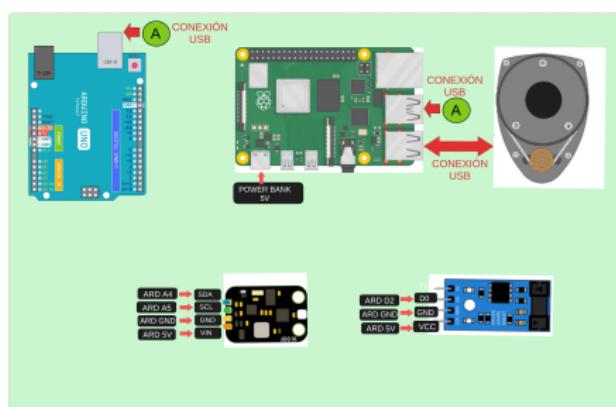


# Hardware [Componentes del Robot]

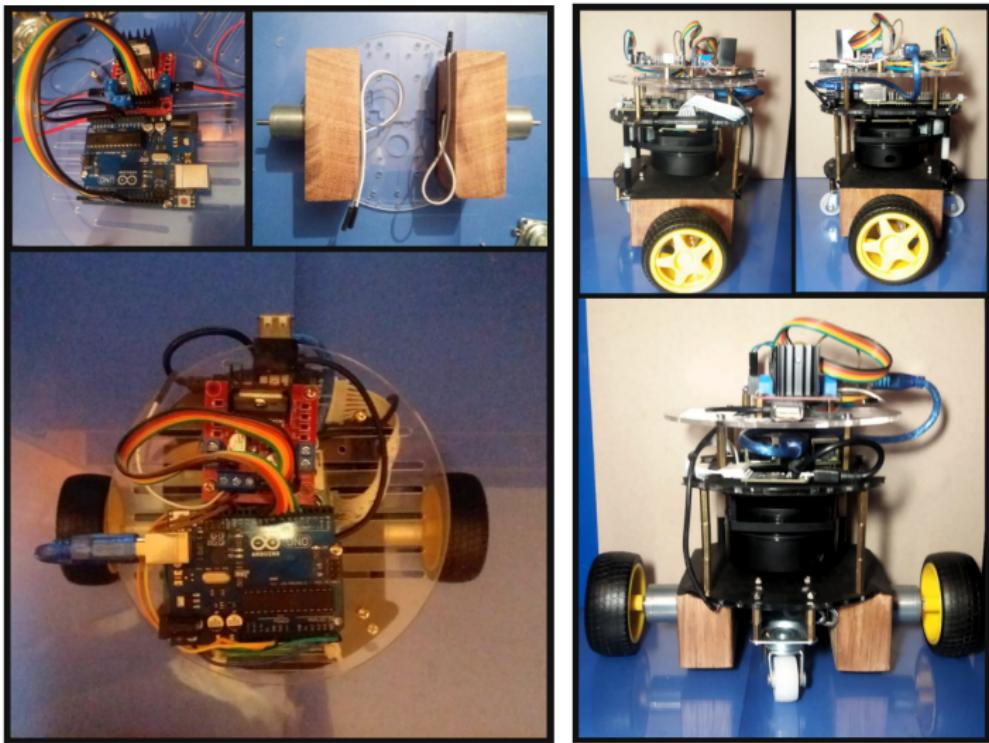
## Zona de Potencia



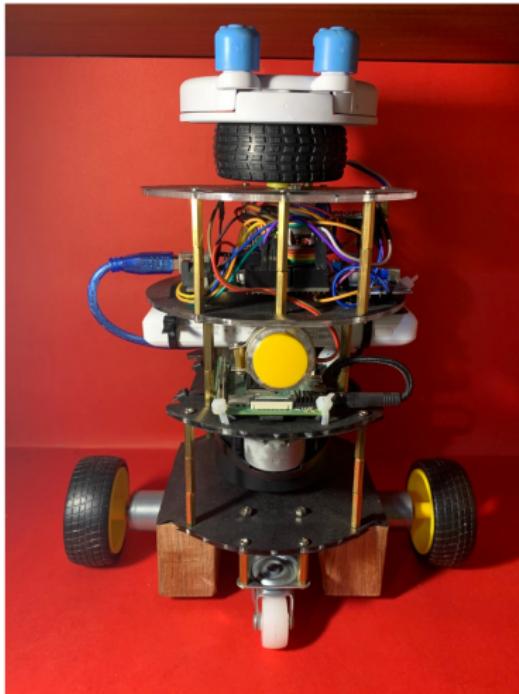
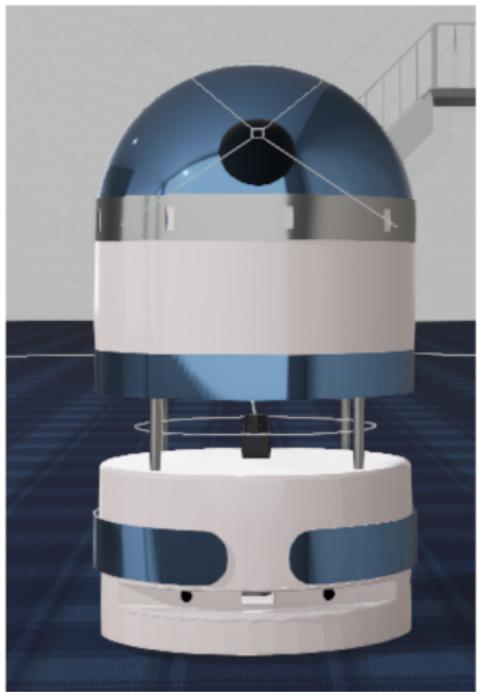
## Zona de Procesamiento



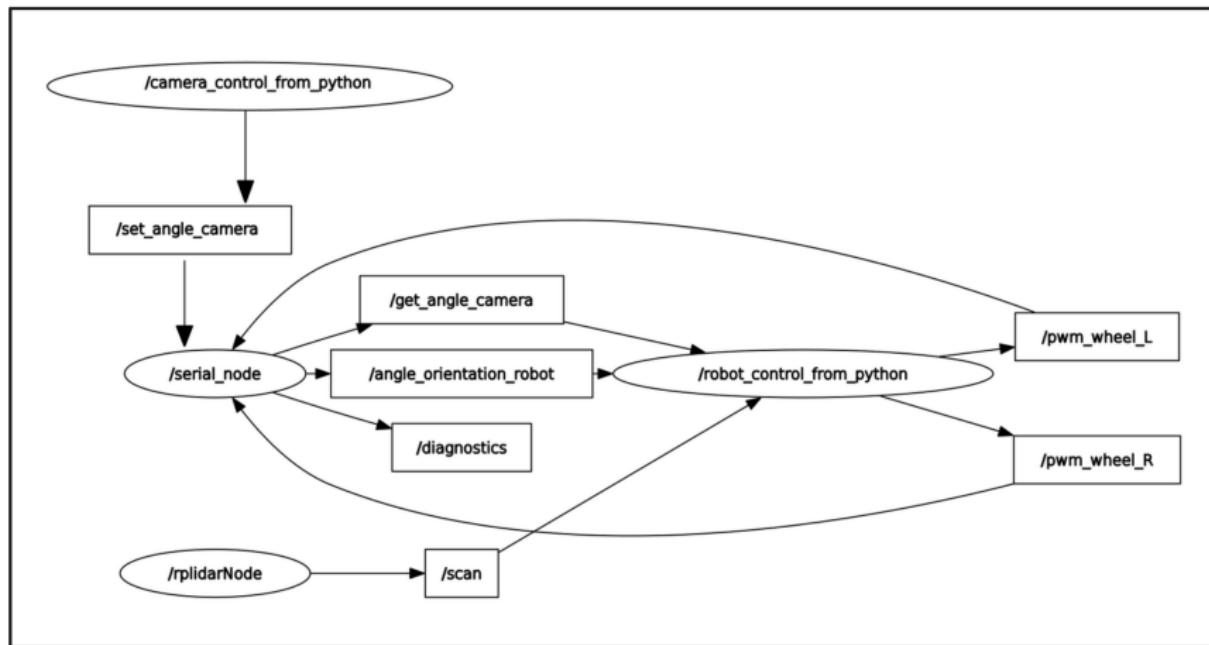
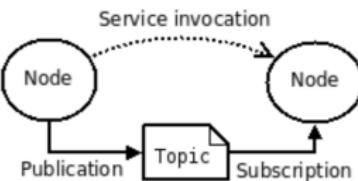
# Hardware [Construcción y Ensamblaje]

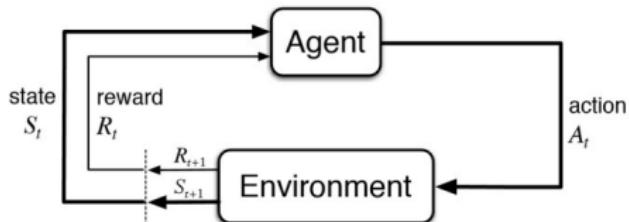


# Hardware [Construcción y Ensamblaje]



# Control de Robot [ROS]





## Retorno

$$G_t = \sum_{i=0}^n \gamma^i R_{i+1}(S_i, a_i); \quad \text{siendo } 0 \leq \gamma \leq 1 \quad (1)$$

## Función de Valor de Estado - Acción (Q value)

$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t | S_t = s, A_t = a]; \quad \forall s \in S, \forall a \in A \quad (2)$$

## Ecuacion de Bellman

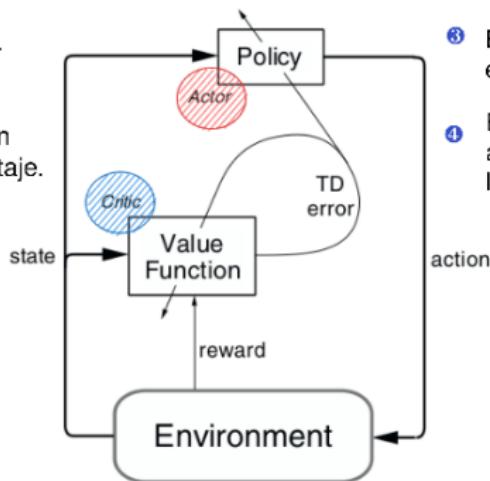
$$Q_\mu(s_t, a_t) = R(s_t, a_t) + \gamma Q_\mu(s_{t+1}, a_{t+1}) \quad (3)$$

## Aprendizaje de Diferencia Temporal (TD ERROR)

$$\delta_t = \underbrace{R(s_t, a_t) + \gamma Q'(s_{t+1}, a_{t+1}) - Q(s_t, a_t)}_{Q'(s_t, a_t), \text{ por Bellman}} \quad (4)$$

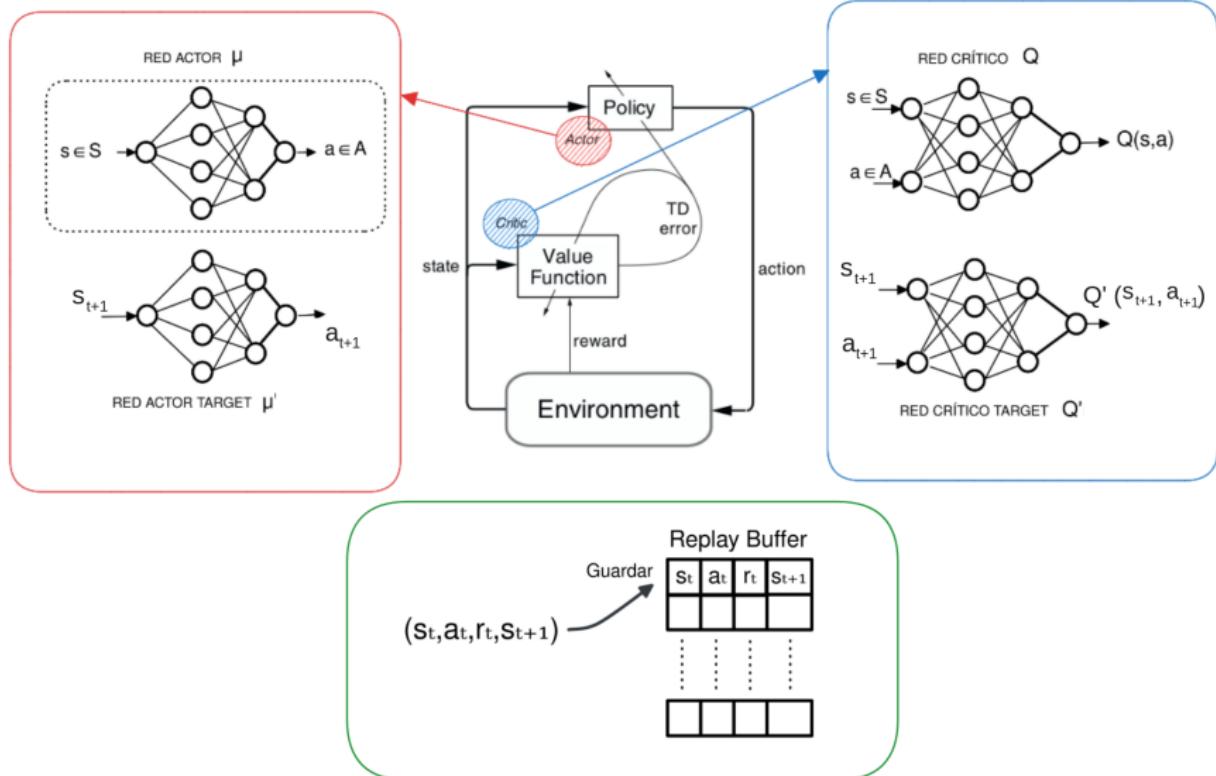
## MODELO ACTOR-CRÍTICO

- ① El actor tomará una acción.
- ② El Crítico evaluará la acción tomada, asignando un puntaje.

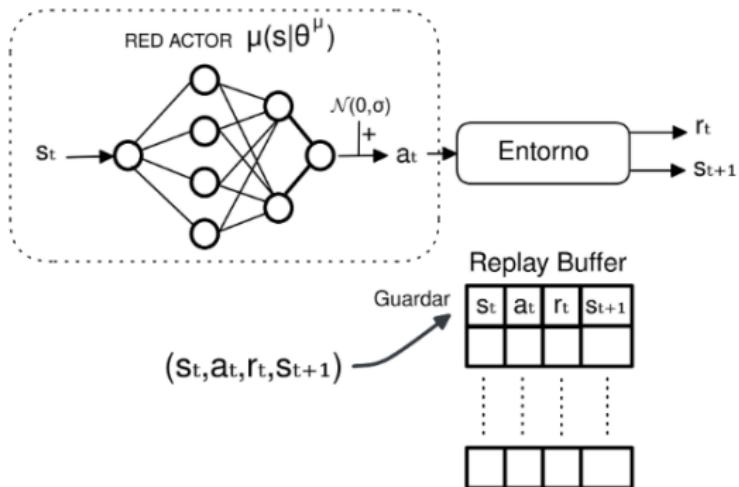


- ③ El Crítico mejorará disminuyendo el TD error.
- ④ El Actor mejorará buscando que sus acciones sean de mayor puntaje en la evaluación del crítico.

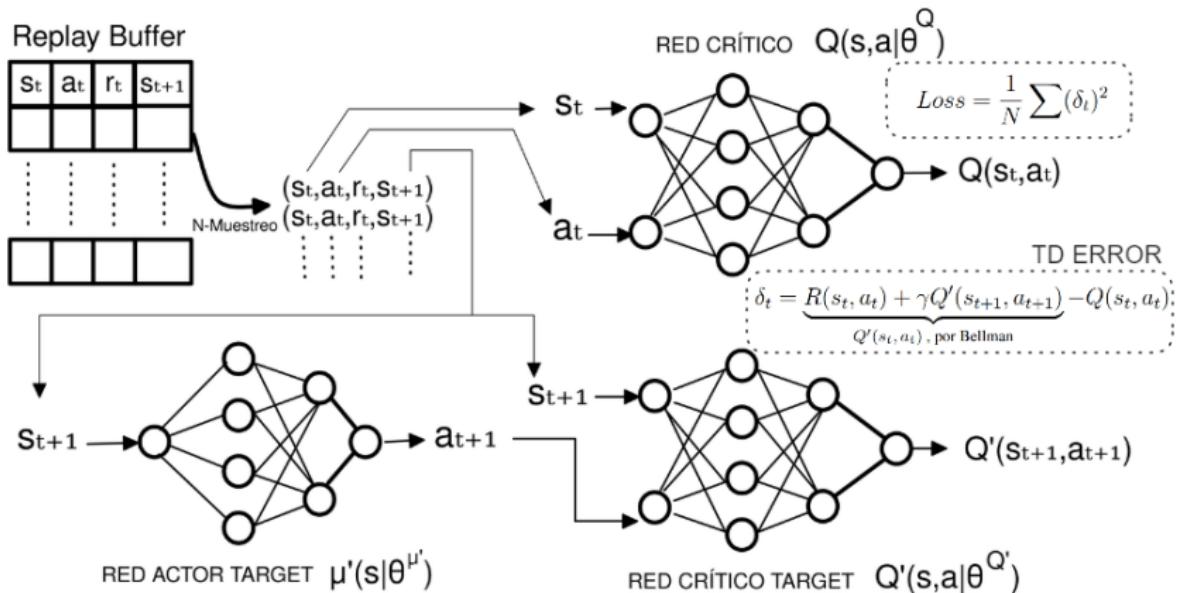
## ARQUITECTURA DDPG



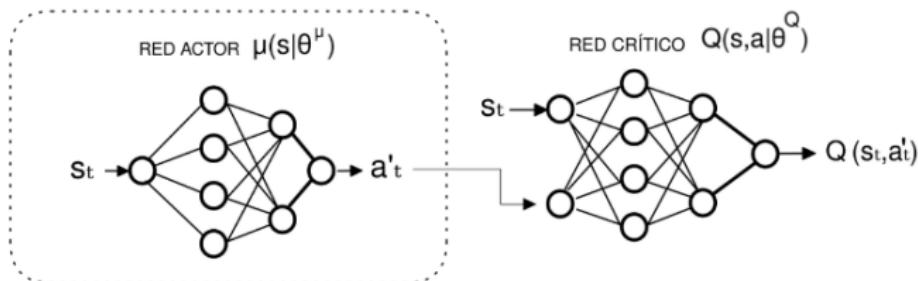
## Paso 1: Llenado de Replay Buffer



## Paso 2: Muestreo, Predicción y mejora de Crítico



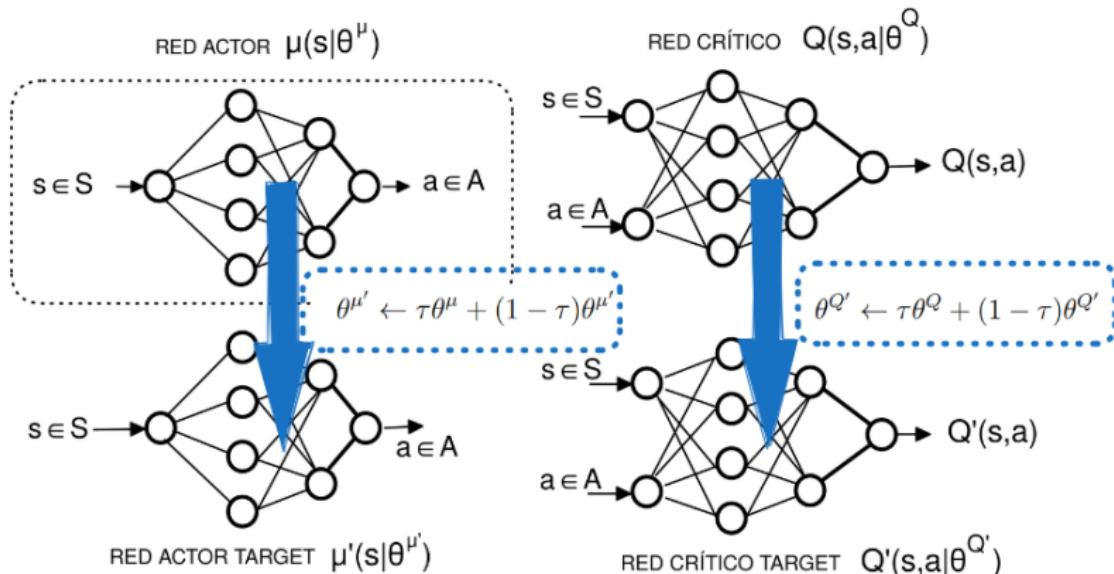
## Paso 3: Mejora de Actor



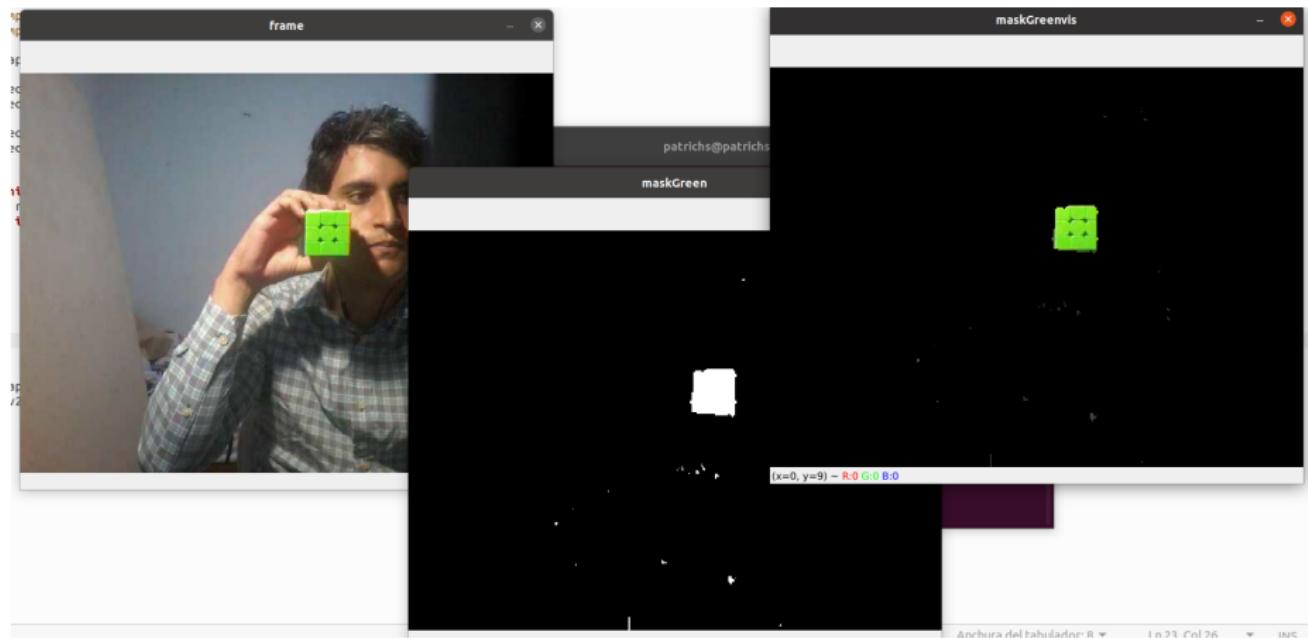
$$J = \frac{1}{N} \left( \sum Q(s_t, a'_t) \right), \text{ donde } a'_t = \mu(s_t | \theta^\mu)$$

$$\nabla J = \frac{1}{N} \sum (\nabla_{a'_t} Q(s_t, a'_t) \nabla_{\theta^\mu} \mu(s_t | \theta^\mu))$$

## Paso 4: Actualización de Redes Target



# Control de Robot [OpenCV]



## FUNCIÓN DE RECOMPENSA

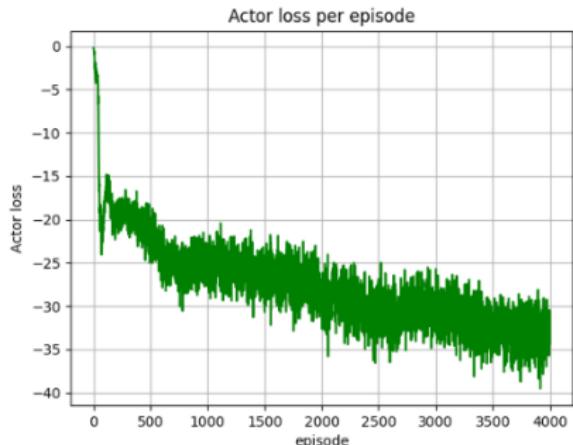
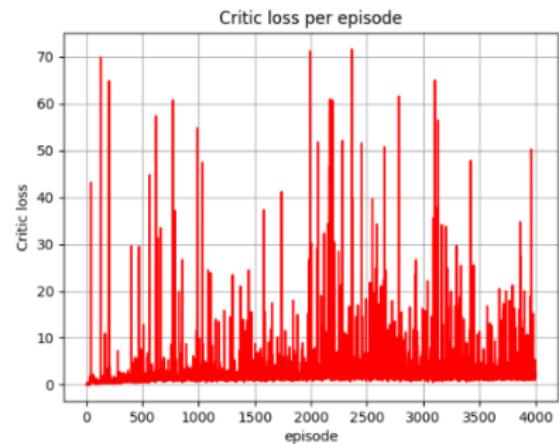
Evento	Recompensa	Tipo	Descripción
Robot Móvil alcanza objetivo	+100	Terminal	Estimula al agente a buscar llegar a la persona objetivo.
Robot móvil se mueve hacia objetivo.	$\text{prev\_dist\_to\_goal} - \text{dist\_to\_goal}$	No Terminal	Estimula al agente avanzar hacia objetivo.
Robot móvil se aleja de objetivo.	-0.001	No Terminal	Evita que el robot quiera retroceder para hacer mas puntos luego.



# Metodología y desarrollo [Entrenamiento del modelo DDPG]

```
1 class HumanFollowingRobotSupervisor(RobotSupervisorEnv):
2     def __init__(self):
3         self.action_space = gym.spaces.box.Box(
4             low=np.array(["69 dimensiones"], dtype=np.float32),
5             high=np.array(["69 dimensiones"], dtype=np.float32))
6         self.observation_space = gym.spaces.box.Box(
7             low=np.array(["2 dimensiones"], dtype=np.float32),
8             high=np.array(["2 dimensiones"], dtype=np.float32))
9
10    def get_observations(self):
11        "Aqui se realiza la obtención de datos de velocidades del
12        robot, angulo de cámara, datos del LIDAR y se normalizan"
13        return [robot_velocity_x, robot_velocity_y,
14                robot_orientation_x, robot_orientation_y,
15                angle_camera] + range_image
16
17    def get_reward(self):
18        dist_to_goal = "Aqui de calcula la distancia a meta"
19        reward = max(self.prev_dist_to_goal - dist_to_goal, 0)*10
20        if self.prev_dist_to_goal - dist_to_goal < 0:
21            reward = -0.001
22        if dist_to_goal < self.dist_to_goal_reached:
23            reward = 100.0
24        self.prev_dist_to_goal = dist_to_goal
25        return reward
26
27    def get_default_observation(self):
28        return np.concatenate([np.array([0.0, 0.0, 1.0, 0.0, 0.0]),
29                             np.array([1.0]*self.lidar_resolution)])
30
31    def is_done(self):
32        if self.prev_dist_to_goal < self.dist_to_goal_reached:
33            print("Goal")
34            self.steps = 0
35            return True
36        #the radious of the robot is 0.265
37        for laser in self.lidar.getRangeImage():
38            if laser < self.dist_to_obstacle_reached and self.train:
39                print("Obstacle")
40                self.steps = 0
41                return True
42        if (abs(self.robot.getPosition()[0]) > 4.0
43            or abs(self.robot.getPosition()[1]) > 4.0) and self.train:
44            print("Arena Limit")
45            return True
46        return False
47
48    def apply_action(self, action):
49        speed_right = float(action[0]) * 4.0
50        speed_left = float(action[1]) * 4.0
51        self.wheels[0].setPosition(float('inf'))
52        self.wheels[0].setVelocity(speed_right)
53        self.wheels[1].setPosition(float('inf'))
54        self.wheels[1].setVelocity(speed_left)
55
56    def setup_motors(self):
57        self.wheels[0] = self.getDevice('wheel_right_joint')
58        self.wheels[1] = self.getDevice('wheel_left_joint')
59        self.motor_camera = self.getDevice('camera_motor')
60        for i in range(len(self.wheels)):
61            self.wheels[i].setPosition(float('inf'))
62            self.wheels[i].setVelocity(0.0)
63            self.motor_camera.setPosition(float('inf'))
64            self.motor_camera.setVelocity(0.0)
```

# Metodología y desarrollo [Resultados del modelo DDPG]



# Metodología y desarrollo [Resultados del modelo DDPG]

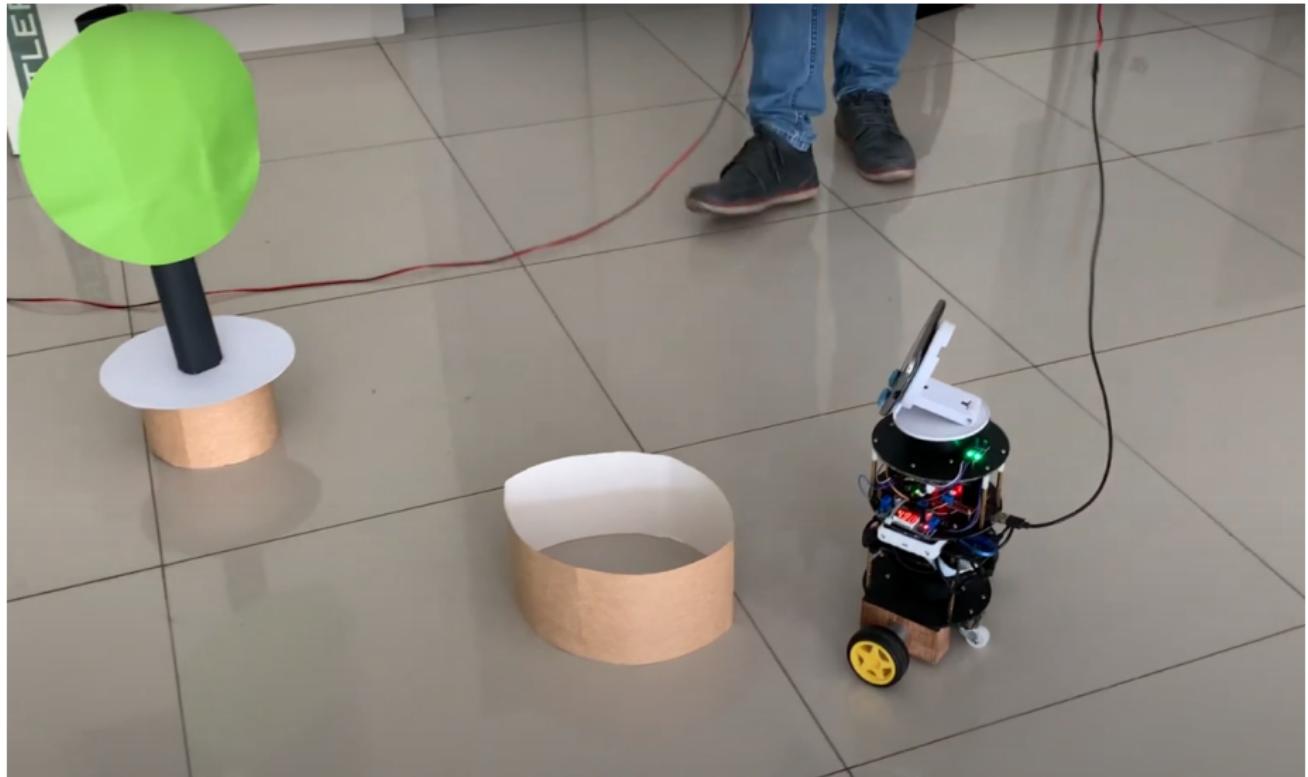


# Resultados de Entrenamiento



[Ver Resultados de Entrenamiento](#)

# Despliegue de modelo en el Robot Real



[Video1](#) [Video2](#)

## Conclusiones

- Se logró construir un Robot físico que fue virtualmente entrenado en un entorno virtual en Webots compatible con OpenAI gym y que cumple con las exigencias que este proyecto amerita para funcionar correctamente en el mundo real.
- Se logró comprender como funcionan los modelos DDPG, destacando su adaptabilidad a datos del mundo real y que contienen ruido.

- Incursión en mejores modelos de Visión Artificial para el seguimiento de Personas.
- Añadiendo mayor dimensionalidad a los Estados.