ISSN 1870-4069

Integración de aprendizaje por refuerzo a simulador físico y análisis de funciones de recompensa para robots humanoides

Alfonso Brown González, Fernando Cossío Ramírez, Edgar Omar López Caudana

Instituto Tecnológico y de Estudios Superiores de Monterrey,
Ciudad de México, México
{abrown.glez,oissoc.ref}@gmail.com, edlopez@tec.mx

Resumen. Se presenta la integración de un entorno de simulación del robot NAO en Webots junto con el entorno de programación para aprendizaje por refuerzo de OpenAI: Gym. Se utilizan funciones de recompensa para medir el desempeño de distintos entornos de aprendizaje y así definir cuáles son las características, variables y parámetros con las que esta función debe contar para llegar a un resultado observable. El proyecto se encuentra en desarrollo por lo que aún no existen resultados definitivos. El presente artículo concluye que hace falta trabajo para alcanzar la meta principal, pero que los avances actuales son observables y medibles. El proyecto final busca crear un sistema de control jerárquico basado en inteligencia artificial y compararlo con los sistemas de control que existen actualmente para el NAO. Por tanto, una vez terminada la primera etapa, se utilizarán dichas funciones de recompensa en un controlador de bajo nivel y se desarrollará el entorno de entrenamiento por RL para tareas de alto nivel.

Palabras clave: aprendizaje de máquina, aprendizaje por refuerzo, NAO.

Reinforcement Learning Framework Integration with Physics Simulator and Reward Function Analysis in Humanoid Robot

Abstract. We present the integration of a NAO robot in Webots with the Reinforcement Learning framework from OpenAI: Gym. We then try different reward functions to measure the ability of a single algorithm to learn, in order to determine the key characteristics, variables, and parameters that this function should have in order to reach observable results. This is a working proyect so the results are not final. This article concludes that there is still work to be done in order to reach the objective, but that actual results are already measurable. On the long term, we intend to design a hierarchical control system based on Reinforcement Learning, and compare it to other control systems currently being used

with the NAO robot. Therefore, once we complete this first stage, the reward functions defined here will be used for a Low Level Controller, and a training environment for a High Level Controller will be designed.

Keywords: machine learning, reinforcement learning, NAO.

1. Introducción

1.1. Problemática y justificación

Poco a poco las redes neuronales y las tendencias de inteligencia artificial se han ido apoderando de áreas en donde la programación tradicional solía reinar. Uno de estos campos es la robótica, donde muchos problemas de control y realización de tareas han empezado a migrar a soluciones más óptimas basadas en IA. Sin embargo, la implementación de estas nuevas tecnologías no siempre es directa ni sencilla. Este trabajo busca presentar una alternativa amigable de integración y evaluación de algoritmos de Aprendizaje por Refuerzo (RL por sus siglas en inglés) en un simulador físico, para luego llevar el aprendizaje y tareas realizadas al robot real. Buscamos desarrollar inicialmente un estándar de programación a seguir para poder integrar *toolkits* y APIs de alto nivel (como Gym, Baselines, Keras, entre otros) que simplifiquen el proceso de diseñar y evaluar algoritmos de aprendizaje en un simulador físico como Webots.

1.2. Objetivos

El objetivo general del proyecto es: "Desarrollar un estándar de programación y entorno de simulación de robots basado en RL que permite entrenar tareas definidas por una función de recompensa y evaluar el desempeño del aprendizaje y los algoritmos utilizados."

Específicamente, se plantean los siguientes objetivos:

- Desarrollar un ambiente de simulación del robot humanoide NAO, utilizando el framework para Aprendizaje por Refuerzo de OpenAI, Gym.
- Diseñar y probar funciones de recompensa para lograr tareas de bajo nivel[1] y medir las variables de estado claves en conseguir que el robot aprenda a resolver la tarea de manera eficiente.
- Evaluar diferentes algoritmos de RL, e hiperparámetros de los mismos, según su eficiencia y capacidad para converger a modelos que solucionan las tareas propuestas.

2. Marco teórico

2.1. Aprendizaje por refuerzo

El término de Aprendizaje por Refuerzo, RL, es una de las tres grandes áreas del Aprendizaje de Máquina (*Machine Learning*) junto con el Aprendizaje

[1] Como mantener el balance y dar pasos para caminar

supervisado y el Aprendizaje no supervisado. Surgió como una solución a los procesos de decisiones de Markov (MDP por sus siglas en inglés). Gráficamente un MDP se puede representar como una cadena de Markov, como se muestra en la figura 1.

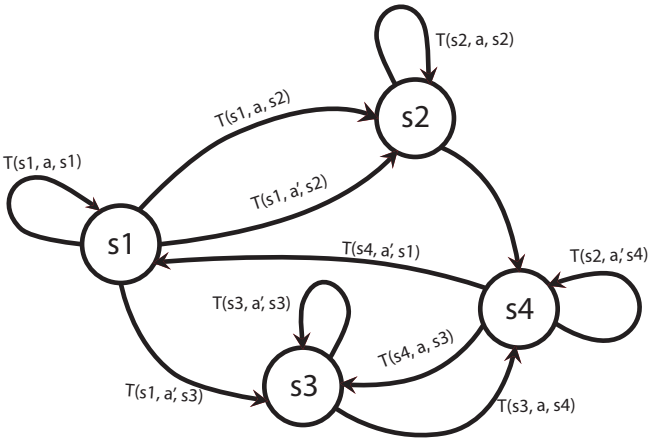

Fig. 1. Ejemplo de la representación gráfica de un proceso de Markov sencillo con 4 estados (s1, s2, s3, s4) y dos acciones (a, a'). La función T representa la probabilidad de transición de un estado a otro dada una acción. A cada flecha de transición de estado también le corresponde una recompensa.

Estos procesos cuentan con un espacio de estados S y un espacio de acciones A y cumplen con algunas características, como que existe una probabilidad $T(s, \pi(s), s')$ de transicionar del estado $s \in S$ al estado s' dada una acción $a \in A$ determinada por la política $\pi(s)$. Entonces, para cada s está definida una política $\pi(s)$ que determina la acción a tomar. Además, en cada transición existe una función de recompensa $R(s, \pi(s))$, que se busca maximizar. De todo esto surgen los valores V, definidos por la ecuación de Bellman como:

$$V^{\pi}(s) = R(s, \pi(s)) + \gamma \sum_{s'} T(s, \pi(s), s') V^{\pi}(s'), \tag{1}$$

donde γ es el factor de descuento, que sirve para dar preferencia a las recompensas obtenidas en la menor cantidad de transiciones (o en el menor tiempo) posibles.

Sin embargo, resolver un MDP por valores de estado V se vuelve en extremo tedioso y complicado, sobre todo porque en aplicaciones reales la primera vez que un agente es sometido a un entorno, no se conoce por completo el espacio de estados, ni mucho menos la probabilidad de transición entre ellos. De esto surgió el algoritmo de Q-Learning [12], el cual modifica la ecuación de Bellman para encontrar valores Q de cada par acción-estado sin necesidad de conocer la

función T. Estos valores Q se calculan por el siguiente método iterativo:

$$Q(s_t, a_t) \leftarrow (1-\alpha)Q(s_t, a_t) + \alpha(r_t + \gamma \max_a(Q(s_{t+1}, a)), \tag{2}$$

donde α es el factor de aprendizaje, y r_t es la recompensa de la transición en el instante t.

Este algoritmo, junto con sus variantes de aprendizaje profundo que incorporan redes neuronales para calcular los valores Q, es bueno para entornos relativamente simples, y preferiblemente con espacios de estado y acción discretos. Para entornos más sofisticados y continuos, como lo es la simulación de un robot, es necesario recurrir a algoritmos que actúan directamente sobre la política π [5,6,9].

Proximal Policy Optimization (PPO) [10]

En 2017 investigadores de OpenAI desarrollaron un algoritmo que buscaba ser escalable, eficiente y robusto, de tal forma que sin modificar los hiperparámetros se pudieran obtener buenos resultados en distintos entornos, discretos o continuos. Con esta intención surgió PPO que, como todos los algoritmos de gradiente de política, calcula una estimación del gradiente basado en las recompensas y realiza un ascenso estocástico. La ventaja es que además incorpora regiones de confianza (basado en algoritmos de TRPO) [8], que evitan que el ascenso de gradiente se aleje demasiado de la máxima recompensa hasta el momento, para no causar actualizaciones destructivas a la política. Se calcula entonces una pérdida L^{CPI}, donde CPI es la abreviación de "Conservative Policy Iteration" (Actualización de política conservadora), como:

$$L^{CPI}(\theta) = \hat{\mathbb{E}}_t[r_t(\theta)\hat{A}_t], \tag{3}$$

donde $r_t(\theta)$ es la razón de probabilidad $r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$, $\hat{\mathbb{E}}_t$ es la expectativa y $\hat{A}_t$ es una estimación de la ventaja en el tiempo t, basada en las recompensas. Se define además un hiperparámetro ϵ pensado para que $r_t(\theta)$ no se devíe del intervalo $[1-\epsilon, 1+\epsilon]$. La estimación de pérdida con región de confianza utilizando un factor de ajuste (*clip factor*) se calcula como:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)]. \tag{4}$$

Se pueden lograr resultados similares utilizando otros métodos de ajuste para la región de confianza, como penalizaciones según la divergencia KL como describe [10].

Para incorporar además los valores de estado $V(s)$ y asegurar que en la implementación exista suficiente exploración, se introduce un bonus por entropía $S(\pi)$ y una pérdida por los valores de estado, y se obtiene la ecuación completa de pérdida:

$$L_t^{CLIP+VF+S}(\theta) = \hat{\mathbb{E}}_t[L_t^{CLIP}(\theta) - c_1 L_t^{VF}(\theta) + c_2 S[\pi_\theta](s_t)]. \tag{5}$$

Finalmente, el algoritmo queda como:

Algoritmo 1 Proximal Policy Optimization.

para iteración=1,2,... **hacer**
  para actor=1,2,... **hacer**
    Correr la política $\pi_{\theta_{\text{old}}}$ en el entorno por T pasos
    Calcular estimados de ventaja $\hat{A}_1, ..., \hat{A}_T$
  fin para
  Optimizar L con respecto a θ, con K epocas y minibatch tamaño $M <= NT$
  $\theta_{\text{old}} \leftarrow \theta$
fin para

2.2. Esquema de trabajo Gym

Gym[2] es un *framework* open-source de entrenamiento por RL creado por OpenAI. Permite desarrollar y definir entornos por completo, partiendo de los agentes, los estados, las transiciones y sobre todo las funciones de recompensa. Los agentes además son fácilmente integrados con otras herramientas computacionales como Baselines, Keras o TensorFlow para realizar su entrenamiento. Gym establece la estructura general que debe seguir cualquier entorno en cuanto al orden de los directorios, las clases y métodos de Python que el agente utiliza para aprender. Existen más de 2000 entornos de entrenamiento públicos en internet y muchos proveen ejemplos de cómo integrar el entorno a otro software.

Para nuestro proyecto esperamos crear diferentes ambientes de aprendizaje, definiendo funciones de recompensa según la acción a realizar e integrando Gym a un simulador físico en donde podamos observar los resultados. Esta integración hace que no se necesite trabajo ni conocimientos en implementar todos los factores físicos involucrados en la respuesta del entorno y en las recompensas (como la gravedad, fuerzas, torque, etc.).

En alguna etapa posterior, ya que esperamos que este proyecto se vuelva accesible a cualquier persona, proponemos realizar el proceso de aprendizaje en entornos jerárquicos, como se observa en el proyecto DeepLoco[7]. De alguna forma se puede entender que el propósito es usar Gym como backend de nuestro software para un controlador a bajo nivel (LLC), mientras que una interfaz amigable define el controlador de alto nivel (HLC). El LLC se encarga de entrenar y premiar al robot por tareas sencillas, como mantener el equilibrio o dar un paso, mientras que el HLC se encarga de tareas complejas, como resolver un laberinto o patear un objeto.[2]

Los autores en [3] también proponen un sistema jerárquico en cuatro niveles para el robot NAO, basando las recompensas de los controladores a bajo nivel en

[2] Definimos tareas sencillas como problemas de control a bajo nivel, cuya solución consiste en llegar a un estado estable. Tareas complejas se refiere a tareas que dependen de variables no observables por el controlador de bajo nivel y que a menudo involucran la participación de un supervisor externo. En un caso de programación tradicional, el LLC viene implementado en las funciones ya definidas y sistema operativo interno del robot, mientras que el HLC sería el usuario que programa el robot con una finalidad en específico.

el Punto de Momento Cero. Nosotros buscaremos explorar diferentes recompensas, desde alternativas más generales (como el tiempo que se mantiene de pie), hasta otras más específicas basadas en el Centro de Masa o en la desviación de una pose ya definida.

2.3. Simulador Webots y robot NAO

El NAO[11] es un robot diseñado por Aldebaraan Robotics en el 2007, creado como una poderosa herramienta de programación y educación en robótica para niños, adolescentes y adultos jóvenes. Posteriormente se empezó a usar en competencias como el RoboCup y en diferentes aplicaciones académicas, de investigación y de cuidado de salud. Aldebaraan fue comprado en 2013 por una compañía japonesa llamada Softbank Robotics que hasta la fecha son quienes mantienen al robot NAO.

El robot actualmente se encuentra en su sexta iteración. Cuenta con 25 grados de libertad si se toman en cuenta los dedos y 21 si no. Opera con una versión de Linux llamada NAOqi como sistema operativo. Tiene además dos cámaras de alta definición, cuatro micrófonos, dos transmisores/receptores de infrarojo, un sonar, siete sensores de tacto, ocho sensores de presión y sensores de posicionamiento como acelerómetro y giroscopio.

Para trasladar el entrenamiento a un entorno físico y medible donde se puedan observar los resultados, integraremos nuestro entorno de aprendizaje de Gym con el simulador Webots [13]. Escogimos este simulador ya que cuenta con modelos hechos del robot NAO, con todos sus grados de libertad y con los sensores internos ubicados y calibrados de manera que simulan a los del robot real. Webots permite establecer si cada objeto en el entorno es un supervisor o funciona con un controlador. Un supervisor tiene más facultades sobre el entorno que un controlador, lo que nos posibilita reiniciar la posición del robot al final de cada episodio, para que el controlador únicamente se dedique a explorar y aprender a maximizar sus recompensas. En la figura 2 se muestra el robot nao en el simulador Webots.

3. Resultados

3.1. Integración de Webots, Gym y el modelo del robot NAO

Se hicieron pruebas con diferentenes simuladores físicos como PyBullet y MuJoCo. Sin embargo, no existen modelos libres del NAO ni la integración del robot con sus sensores internos. Al utilizar Webots, tenemos el modelo del robot integrado y en los programas de ejemplo se cuentan con ejercicios de calibración de los sensores para que operen de manera similar a los del robot real. Estos valores son los que utilizamos para generar el estado y las observaciones en los que se encuentra el NAO.

Lo primero que se logró es integrar los controles del robot virtual para que se puedan leer los sensores de la tabla 2. También logró leer las posiciones en

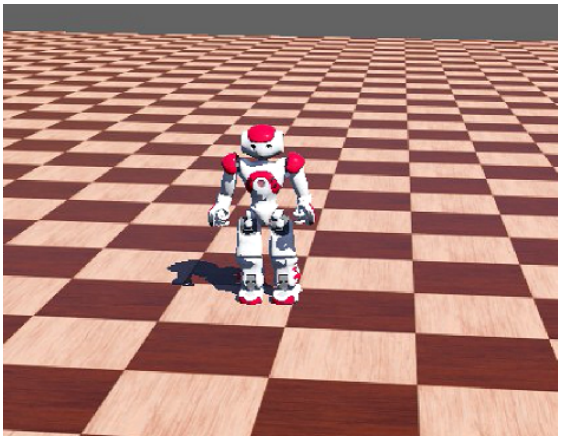

Fig. 2. Robot NAO en simulador Webots.

radianes y configurar la posición objetivo de las articulaciones mencionadas en la tabla 1. Cada articulación se linearizó de acuerdo a su posición angular mínima y máxima para que tanto la lectura como la escritura quedara en el intervalo $[-1.0, 1.0]$. Para obtener la posición del motor en dicho intervalo se utiliza la siguiente ecuación:

$$pos\_linearizada = \frac{pos_{actual} - pos_{min}}{pos_{max} - pos_{min}} * 2 - 1. \tag{6}$$

También se hizo un ajuste en el controlador para que las articulaciones "RHipYawPitch" y "LHipYawPitch" se comporten de manera simétrica, como sucede en el robot real.

Una vez que se logró manejar el robot con código de Python en el simulador, se integró el framework de Gym y se creó el entorno de entrenamieto para poder correr los algorítmos de RL. Luego de lograr la integración exitosamente, es posible utilzar la librería de Stable Baselines [4] para implementar y programar en pocas líneas de código los agentes que aprenderán a controlar el robot. Esta librería ya tiene las arquitecturas y redes neuronales de varios algoritmos implementadas y utiliza TensorFlow como backend, lo que nos permitió utilizar TensorBoard como herramienta de evaluación y depuración. [1]

3.2. Entrenamiento para mantener el balance

Como una primera tarea a lograr establecimos el lograr mantenerse en pie sin información previa de como hacerlo. El algoritmo recibe como estado las posiciones de los motores y todos los sensores mencionados en la tabla 2 y puede tomar acciones con una frecuencia de 10Hz.

Logramos que se entrenaran 1M de timesteps del modelo en cuestión en poco menos de 10 horas, lo que significa que, aproximadamente, el entrenamiento se

Tabla 1. Articulaciones del robot NAO.

HeadYaw	HeadPitch	RShoulderPitch
RShoulderRoll	RElbowYaw	RElbowRoll
RWristYaw	RPhalanx1	RPhalanx2
RPhalanx3	RPhalanx4	RPhalanx5
RPhalanx6	RPhalanx7	RPhalanx8
LShoulderPitch	LShoulderRoll	LElbowYaw
LElbowRoll	LWristYaw	LPhalanx1
LPhalanx2	LPhalanx3	LPhalanx4
LPhalanx5	LPhalanx6	LPhalanx7
LPhalanx8	RHipYawPitch	RHipRoll
RHipPitch	RKneePitch	RAnklePitch
RAnkleRoll	LHipYawPitch	LHipRoll
LHipPitch	LKneePitch	LAnklePitch
LAnkleRoll		

Tabla 2. Sensores del robot NAO.

Acelerómetro	Giroscopio
Unidad Inercial	Sensores de presión
Bumpers	Ultrasónicos

hace con un factor de velocidad de 2.8x. El episodio inicia con una posición de las articulaciones que varía uniformemente en un rango de $\pm 0.25\,\%$ y el episodio termina si el robot cae. Para las pruebas, se congeló todo el tren superior (Cabeza, cuello, brazos) para reducir la complejidad del problema a solo 11 grados de libertad y lograr converger a un resultado en una cantidad menor de iteraciones.

Se utilizó el algorítmo PPO2 de Stable Baselines[3] con los parámetros por defecto para hacer pruebas preliminares y demostrar que el robot efectivamente está aprendiendo.

Primero definimos una función de recompensa muy general, donde cada 100 ms se otorgan 5 puntos simplemente por el hecho de que no se haya caído, sumado a la altura a la que el robot se encuentra en ese momento. Con esta función de recompensa, después de 40k pasos se llegó a un comportamiento poco estable en donde el robot brincaba para tratar de obtener la mayor altura, pero en promedio solo duraba 1.2 segundos de pie.

Posteriormente, se ajustó la función para descontar puntos por tener demasiada aceleración, demasiada velocidad, por utilizar torque (para reducir el consumo de energía y obtener movimientos más suaves) y por tener valores de gravedad en otro eje que no sea $-z$.

Específicamente se utilizó la siguiente función de recompensa r:

[3] Algoritmo descrito anteriormente, optimizado para usar GPU en un ambiente vectorizado, contra PPO1 de la misma librería que usa MPI, ya deprecado.

$$r = 2 - d_{torque} - d_{height} - d_{accel} - d_{vel} - d_{distance} - d_{pose}, \tag{7}$$

donde d denota el descuento calculado a partir de las siguientes ecuaciones (8-13):

$$d_{torque} = \frac{\sum_j \text{abs}(torque_j)}{60}, \tag{8}$$

$$d_{height} = (\text{abs}(z - 0.33) * 10)^2, \tag{9}$$

$$d_{accel} = \text{abs}(a_x) + \text{abs}(a_y) + \text{abs}(a_z + 9.81), \tag{10}$$

$$d_{vel} = \text{abs}(v_x) + \text{abs}(v_y) + \text{abs}(v_z), \tag{11}$$

$$d_{distance} = \sqrt{(x^2 + z^2)}, \tag{12}$$

$$d_{pose} = \frac{\sum_j \text{abs}(position_j - position'_j)}{12}. \tag{13}$$

Después de aproximadamente 100k pasos de simulación el robot aprendió a tirarse al piso lo más rápido posible para recibir el puntaje menos negativo posible. Ya que los descuentos resultaban siempre más grandes que la recompensa de mantenerse de pie. Para tratar de compensarlo, se modificó la función de la siguiente manera:

$$r = 5 - w_1 d_{torque} - w_2 d_{height} - w_3 d_{accel} - w_4 d_{vel} - w_5 d_{distance} - w_6 d_{pose}, \tag{14}$$

donde $[w_1, w_2, w_3, w_4, w_5, w_6]$ tomaron los valores $[0.5, 2, 1, 1, 0.5, 1]$.

Logramos que el robot se estabilizara por un par de segundos y después de 4M de iteraciones obtuvo una recompensa promedio por episodio de 42.6 y con una duración de pie promedio de 2.1 segundos. Bajo estas mismas recompensas, antes de entrenar la red, el modelo, que era aleatorio, tenía un promedio de recompensa por episodio de -8.35 puntos y una duración de 290 ms.

Además, para probar la robustez del algoritmo, se probaron diferentes hiper-parámetros. Un ejemplo de dos tamaños de red neuronal sobre la misma función de recompensa entrenada por un millón de episodios se puede ver en la figura 3. Se observa que, aunque aprenden con diferentes oscilaciones y ruido, al final la recompensa promedio es muy similar. De igual manera, todos los hiper-parámetros que se probaron no resultaron en diferencias en las recompensas estadísticamente significativas.

Por último, cambiamos la función de recompensa a que fuera más restrictiva, específica y que obligara al robot a encontrar una pose estable. Para esto definimos la pose de un paso izquierdo, ajustando ligeramente las posiciones de los motores hasta obtener un estado estable donde el centro de masa está sobre el

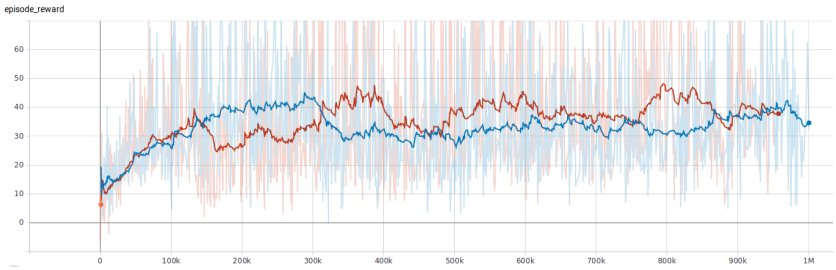

Fig. 3. Comparación de actuación de dos redes neuronales de diferentes tamaños sobre la misma función de recompensa.

eje z. Basándonos en el LLC implementado en DeepLoco[7], definimos la función de recompensa como:

$$r = -w_1 d_{torque} - w_2 d_{height} - w_3 d_{distance} - w_4 d_{pose}, \tag{15}$$

donde los pesos se ajustaron a $[w_1, w_2, w_3, w_4] = [0.1, 0.3, 0.1, 0.5]$ y el descuento por la posición esperada se modificó a:

$$d_{pose} = e^{-(\sum_j \text{abs}(position_j - position'_j))^2}. \tag{16}$$

Después de 400k episodios, el NAO logró una recompensa promedio por episodio de 7.42, y logró mantenerse de pie hasta 5 segundos. En la figura 4 se puede ver la diferencia entre las recompensas promedio del algoritmo sin entrenar y el agente entrenado. Así mismo, en la figura 5 se puede ver la posición semi-estable que alcanzó el robot después de esta etapa de entrenamiento.

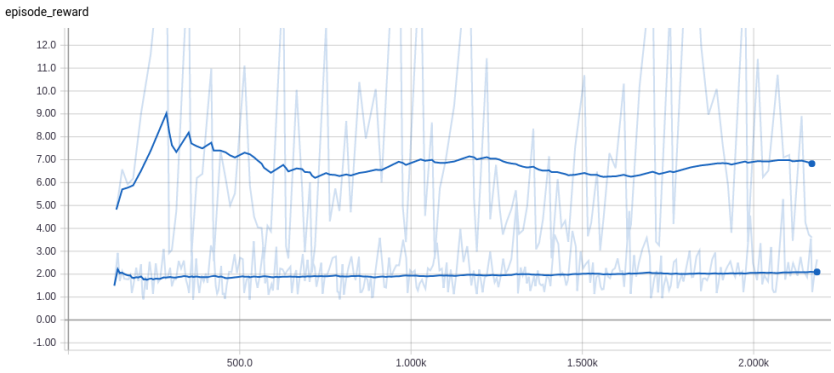

Fig. 4. Comparación de recompensas promedio por episodio entre agente sin entrenamiento y agente entrenado.

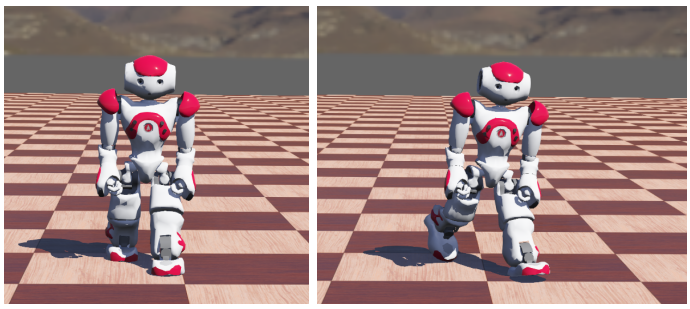

Fig. 5. Del lado izquierdo, la pose objetivo con la que se planteó el descuento. A la derecha el robot entrenado después de 400k episodios. En este episodio se mantuvo de pie y sin moverse por más de 4 segundos.

4. Conclusiones

4.1. Conclusiones del entrenamiento de prueba

En un análisis cualitativo de los resultados, se puede observar que el robot no cumple con la tarea de manera existosa aún. Sin embargo, sí se puede apreciar que el robot aprende a moverse de manera suave gracias al castigo por usar torque, que logra mantener el balance por mayor tiempo y se observa que está aumentando las recompensas promedio de cada episodio. Esto muestra que de alguna manera ha aprendido a mantenerse de pie por un breve intervalo de tiempo a pesar de las diferentes funciones de recompensa. Sin embargo, no hemos encontrado todos los factores más importantes a incluir para que el robot llegue a un estado estable, por lo que se cae después de unos segundos.

Al evaluar las pruebas realizadas, se puede concluir que:

- Para lograr una tarea continua (como mantenerse de pie), el entorno no debe dar una recompensa o castigo escaso, es decir, en el estado terminal del episodio, ya que se comporta como ruido.
- Vivir un timestep más debe ser estocásticamente mejor en recompensa que aventarse al suelo. Dicho en otras palabras, los castigos deben de ser, en promedio, no mayores a la recompensa de seguir de pie. Ya que de lo contrario, el algorítmo converge a terminar el episodio tan pronto como sea posible para evitar perder más puntos.
- Mantener el centro de masa y la aceleración en el eje z es indispensable para que el robot se mantenga de pie, por lo que la función de recompensa debe dar más peso a esto.

4.2. Trabajo a futuro

De acuerdo a los objetivos específicos del proyecto, solo se hemos cumplido por completo con el primero, aunque se ha progresado con el segundo

y tercero. El siguiente paso es seguir explorando recompensas y entrenando el robot para posteriormente establecer los estándares de integración de RL con el NAO y poder evaluar distintos algoritmos. Posteriormente, para poder entrenar las siguientes tareas de un carácter más complejo y comparar resultados contra la programación tradicional, debemos implementar la red neuronal, las recompensas y el algoritmo del controlador de alto nivel.

La velocidad de entrenamiento también es un reto a mejorar, puesto que se requieren tiempos de entrenamiento de al menos 1M de timesteps para lograr resultados similares a los de los humanoides de Roboschool (entorno ejemplo de Gym), que tienen 16 grados de libertad. Para esto, proponemos hacer uso de procesamiento en paralelo en una o varias computadoras.

Referencias

1. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: Large-scale machine learning on heterogeneous systems (2015), https://www.tensorflow.org/, último acceso: 2019/05/11
2. Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., Zaremba, W.: Openai gym (2016)
3. Gil, C., Calvo, H., Sossa, H.: Learning an efficient gait cycle of a biped robot based on reinforcement learning and artificial neural networks. Applied Sciences 9(3), 502 (Feb 2019), https://doi.org/10.3390/app9030502, último acceso: 2019/05/11
4. Hill, A., Raffin, A., Ernestus, M., Gleave, A., Traore, R., Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., Wu, Y.: Stable baselines. https://github.com/hill-a/stable-baselines (2018), último acceso: 2019/05/11
5. Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., Wierstra, D.: Continuous control with deep reinforcement learning (2015)
6. Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., Riedmiller, M.: Playing atari with deep reinforcement learning (2013)
7. Peng, X.B., Berseth, G., Yin, K., van de Panne, M.: Deeploco: Dynamic locomotion skills using hierarchical deep reinforcement learning. ACM Transactions on Graphics (Proc. SIGGRAPH 2017) 36(4) (2017)
8. Schulman, J., Levine, S., Moritz, P., Jordan, M.I., Abbeel, P.: Trust region policy optimization (2015)
9. Schulman, J., Moritz, P., Levine, S., Jordan, M., Abbeel, P.: High-dimensional continuous control using generalized advantage estimation (2015)
10. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., Klimov, O.: Proximal policy optimization algorithms (2017)
11. SoftbankRobotics: NAO the humanoid robot (2018), https://www.softbankrobotics.com/emea/en/nao, último acceso: 2019/05/11
12. Watkins, C.J.C.H., Dayan, P.: Q-learning. Machine Learning 8(3-4), 279–292 (May 1992), https://doi.org/10.1007/bf00992698

13. Webots: http://www.cyberbotics.com, http://www.cyberbotics.com, commercial Mobile Robot Simulation Software