

# Deep RL Arm Manipulation

Miguel A. Colmenares Barboza

**Abstract** — El proyecto presentado en este documento se centran en implementar un agente de Deep Q-Learning Network (DQN) desarrollado por Dustin Franklin en el proyecto "Jetson-Reinforcing" de Nvidia y definir funciones de recompensa para enseñar a un brazo robótico a llevar a cabo dos objetivos principales:

- Haga que cualquier parte del brazo del robot toque el objeto de interés, con al menos un 90% de precisión y un mínimo de 100 carreras.
- Haga que solo la base de la pinza del brazo del robot toque el objeto, con al menos un 80% de precisión para un mínimo de 100 carreras.

Ambos objetivos se logran ajustando los valores de recompensas y ciertos parámetros en el código..

**Index Terms** — Robotics, IEEEtran, Reinforcement Learning, DQN.

## I. INTRODUCCIÓN

El aprendizaje de refuerzo profundo está a la vanguardia de la investigación de inteligencia artificial. Muchos expertos lo ven como un camino hacia la Inteligencia General Artificial. En este proyecto, se crea un agentes artificialmente inteligentes que aprende de la interacción con su entorno, la recopilación de experiencia y un sistema de recompensas con un profundo aprendizaje de refuerzo (RL profundo). Utilizando redes neuronales de extremo a extremo que traducen píxeles brutos en acciones, el agente formado en RL será capaz de exhibir comportamientos intuitivos y realizar tareas complejas.

## II. FUNCIONES DE RECOMPENSAS.

La configuración o programación más relevante se centra en el archivo `ArmPlugin.cpp` ubicado en la carpeta `/gazebo`. El archivo `ArmPlugin.cpp` es un complemento al cual recurre el modelo del brazo robótico mostrado en la simulación. Este complemento es responsable de crear el agente DQN y entrenarlo para aprender a tocar el accesorio. El archivo de objeto compartido del complemento `gazebo libgazeboArmPlugin.so`, adjunto al modelo de robot en `gazebo-arm.world`, es responsable de integrar el entorno de simulación con el agente RL.

En el archivo `ArmPlugin.cpp` se programan una serie de tareas, expuestas a continuación.

### \* Suscribirse a los temas de cámara y colisión:

Los nodos correspondientes a cada uno de los suscriptores ya han sido definidos e inicializados. Solo se crean los suscriptores en la función `ArmPlugin::Load()`.

```
cameraSub = cameraNode -> Subscribe
("/gazebo/arm_world/camera/link/camera/image",
&ArmPlugin::onCameraMsg, this);

collisionSub = collisionNode -> Subscribe
("/gazebo/arm_world/tube/tube_link/my_contact",
&ArmPlugin::onCollisionMsg, this);
```

### \* Crear el agente DQN:

Consultando las instrucciones de la API para crear el agente utilizando la función `Create ()` de la clase `dqnAgent`, en la función `ArmPlugin::createAgent()`.

```
agent = dqnAgent::Create(INPUT_WIDTH,
                         INPUT_HEIGHT, INPUT_CHANNELS,
                         NUM_ACTIONS, OPTIMIZER,
                         LEARNING_RATE, REPLAY_MEMORY,
                         BATCH_SIZE, GAMMA, EPS_START,
                         EPS_END, EPS_DECAY, USE_LSTM,
                         LSTM_SIZE, ALLOW_RANDOM, DEBUG);
```

### \* Definir un control basado en la posición de las articulaciones del brazo:

la salida del DQN asigna una acción particular, que, para este proyecto, es el control de cada articulación para el brazo robótico. En `ArmPlugin::updateAgent()`, hay dos enfoques existentes para controlar los movimientos conjuntos, un enfoque basado en la velocidad de las articulaciones del brazo y otro enfoque basado en la posición de las articulaciones del brazo.

En este proyecto se eligió usar el enfoque basado en la posición. La posición es un valor que simplemente cambia y la velocidad es un valor que varía en base al cero y el cambio del valor es más frecuente y puede ser difícil de capturar. Por esta razón, el control basado en la posición supone ser una solución simple.

```
float joint = (action % 2 == 0) ? ref[action/2] +
actionJointDelta : ref[action/2] - actionJointDelta;

if( joint < JOINT_MIN )
    joint = JOINT_MIN; // #define JOINT_MIN -0.75f
if( joint > JOINT_MAX )
    joint = JOINT_MAX; // #define JOINT_MAX 2.0 f
ref[action/2] = joint;
```

El siguiente conjunto de tareas se basa en la creación y asignación de funciones de recompensa en función de los objetivos requeridos.

#### \*Recompensa: Si la pinza del robot golpea el suelo.

Hay una función llamada `GetBoundingBox()` que devuelve los valores mínimo y máximo de un cuadro que define ese objeto/modelo particular correspondiente a los ejes x, y y z.

Usando esta función, se verifica si la pinza está tocando el suelo o no, y se asigna una recompensa.

```
// Get the bounding box for the gripper
const math::Box& gripBBox = gripper->GetBoundingBox();
const float groundContact = 0.05f;

bool checkGroundContact = (gripBBox.min.z <=
                           groundContact) ? true : false;

if(checkGroundContact)
{ rewardHistory = REWARD_LOSS * 20;
  newReward     = true;
  endEpisode    = true; }
```

#### \* Recompensa provisional basada en la distancia al objeto.

Una función llamada `BoxDistance ()` calcula la distancia entre dos cuadros delimitadores. Usando esta función, se calcula la distancia entre el brazo y el objeto. Luego, se usa esta distancia para calcular una recompensa.

```
if(!checkGroundContact)
{
const float distGoal = BoxDistance
                      (gripBBox, propBBox);

if( episodeFrames > 1 )
{
  const float distDelta =
    lastGoalDistance - distGoal;
  const float alpha = 0.2f;
  // 0.1f; [80%] : 0.2f; [90%]

// Compute the smoothed moving average of the delta of
the distance to the goal

avgGoalDelta = (avgGoalDelta * alpha) + (distDelta *
(1.0f - alpha));

// Compute the smoothed moving average of the delta of
the distance to the goal

  if (avgGoalDelta > 0)
  {
    if(distGoal > 0.0f){
      rewardHistory = REWARD_WIN * avgGoalDelta;}
    else if (distGoal == 0.0f){
      rewardHistory = REWARD_WIN * 10.0f;}
    }
  else {
    rewardHistory = REWARD_LOSS * distGoal;
    }
  newReward = true;
}

lastGoalDistance = distGoal;
}
```

**\* Recompensa basada en la colisión entre el brazo y el objeto.** La función de devolución de llamada `onCollisionMsg`, puede verificar ciertas colisiones. Específicamente, definirá una condición de verificación para comparar si los enlaces particulares del brazo con sus elementos de colisión definidos están colisionando con el `COLLISION_ITEM` o no.

#### //Objetivo [90%]

```
bool collisionCheck =
((strcmp(contacts->contact(i).collision1().c_str(),
        COLLISION_ITEM) == 0) ||
(strcmp(contacts->contact(i).collision2().c_str(),
        COLLISION_ARM) == 0)) ? true : false;
```

#### //Objetivo [80%]

```
bool collisionCheck =
(strcmp(contacts->contact(i).collision2().c_str(),
        COLLISION_POINT) == 0) ? true : false;
```

Dependiendo del objetivo que se esté buscando se usa uno de los dos chequeos de colisión. Luego de determinar si el brazo colisionó con el objeto, se asigna una recompensa.

```
if (collisionCheck){

  rewardHistory = REWARD_WIN * 10;
  // REWARD_WIN*10 [90%] : REWARD_WIN*100 [80%];

  newReward = true;
  endEpisode = true;
  return;
}
```

### III. HIPERPARAMETROS.

Una lista de hiperparámetros se proporciona en el archivo `ArmPlugin.cpp`, en la parte superior. Los hiperparámetros siguientes se configuran para obtener la precisión requerida para el objetivo [80%] y [90%].

#### INPUT\_WIDTH 64 y INPUT\_HEIGHT 64:

Se reduce el tamaño de la imagen capturada por la cámara.

#### OPTIMIZER "RMSprop":

Este tipo de optimizador es una buena opción para redes neuronales recurrentes.

#### LEARNING\_RATE 0.0f - [80%] 0.15f , [90%] 0.2f:

Este parámetro determina qué tan rápido el brazo aprende a tocar el objeto.

#### REPLAY\_MEMORY 10000:

10000 Repeticiones de la experiencia permiten que el agente aprenda de memorias anteriores, lo cual, acelera el aprendizaje y romper correlaciones temporales indeseables.

**BATCH\_SIZE - [80%] 32 , [90%] 16:**

Define el número de muestras que se propagan a través de la red.

**USE\_LSTM true:**

Define la decisión de usar **LSTM** (Long short-term memory).

**LSTM\_SIZE 256:**

Define el tamaño de la unidad de memoria.

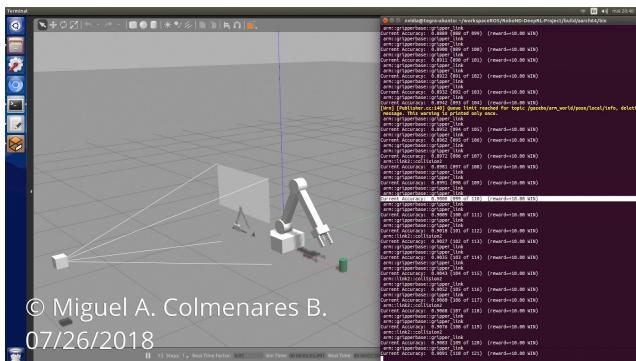
**Define Reward Parameters:**

**REWARD\_WIN 1.0f - REWARD\_LOSS -1.0f:**

Por convención se trabajó con estos valores de recompensa.

**IV RESULTADOS.****Objetivo 1.**

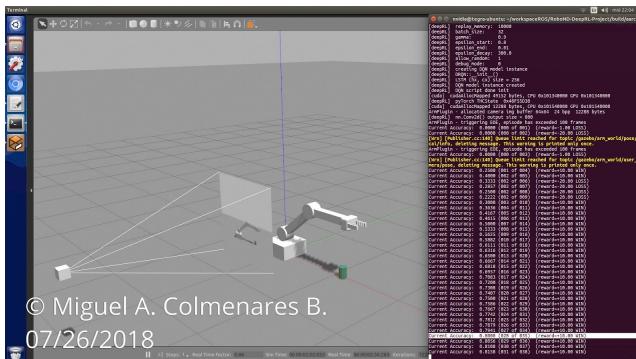
En este objetivo, el brazo robótico debe tocar un objeto (cilindro verde) con un precisión del 90% en al menos 100 carreras. A continuación se muestran los resultados exitosos:



En la carrera número 110, luego de tocar con éxito el objeto en 99 ocasiones, se obtuvo una precisión del 90.00%.

**Objetivo 2.**

En este objetivo, la pinza del brazo robótico debe tocar un objeto (verde cilindro) con una precisión del 80% en al menos 100 carreras. A continuación se muestran los resultados exitosos:



En la carrera número 35, luego de tocar con éxito el objeto en 28 ocasiones, se obtuvo una precisión del 80.00%.

En la ejecución de la prueba del objetivo 1, el agente tardó mucho tiempo en aprender y en el archivo TestGoal90.txt se puede verificar que el objeto fue tocado por varias partes del brazo robótico. Entre ambas pruebas, se puede observar que el agente aprendió más rápido para el objetivo 2 y que el objeto solo fue tocado por gripper\_link. El objetivo 1 continuó hasta que la carrera 300 y el objetivo 2 continuaron hasta la carrera 210 con los resultados "WIN".

**V TRABAJO FUTURO.**

Este proyecto fue muy interesante y es un área de redes neuronales que atrae mi atención. En esencia, los objetivos se lograron con la precisión requerida pero aún hay mucho trabajo por hacer.

En ambas pruebas fue presente un error de simulación, donde el brazo tocó el objeto pero su resultado fue capturado como "LOSS". Creo que este problema se da a razón que el brazo intenta tocar el objeto con un cambio de posición muy abrupto. Para este proyecto se uso solo control de posición y puede que usar un control de velocidad resuelva ese problema. Otra solución podría ser, desarrollar mas la sección de colisión del cilindro verde en el archivo gazebo-arm.world.

Para el objetivo 2 el agente aprende más rápido pero el código se debe pulir más. Por ejemplo, el agente debería dejar de aprender luego de cierta cantidad de carreras exitosas y usar la secuencia de acciones más eficiente, ya que en este caso el objeto mantiene sus características y posición. Podría volver a hacer una prueba aleatoria y volver a prender en un momento aleatorio, pero inclusive esto se debe hacer un momento limitado de veces y solo volver a aprender si se consigue un número importante de resultados "LOSS".

Como trabajo a un futuro más largo, está la situación de cambiar la lata de posición y que el brazo deba buscar la lata en un rango de 180G y 360G. Otro trabajo es que solo el "Gripper Middle" toque la lata con una precisión de 90%.

Con estos trabajos desarrollados considero sería un código completo para el brazo robótico.

Además de este proyecto, quiero desarrollar otros ejemplos del repositorio de Nvidia y seguir investigando sobre el aprendizaje reforzado.