Reinforcement Learning

Youssef Khaky
Robotics Nanodegree
Udacity

***Abstract*—In this project we train a robotic arm to touch a target using Deep Reinforcement learning by using the Q-Learning technique.**

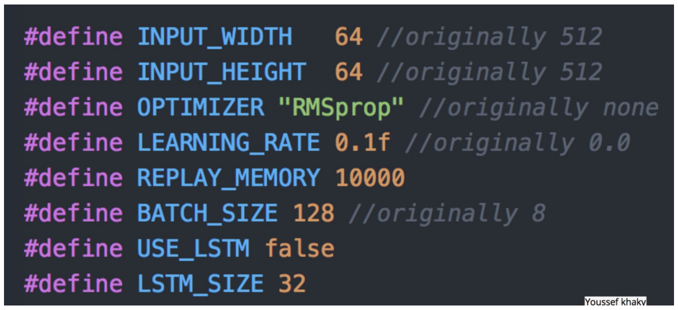
***Index Terms*—Reinforcement Learning, Robotic Arm**

I. Introduction

A Gazebo plugin is attached to a DQN. The plugin is located in the gazebo/ArmPlugin.cpp. A DQN was constructed using the tuned hyper parameters and rewards were set. The agent was left to be trained for some time after which it was able to execute the task with high accuracy.

II. Hyper-parameters Used

The hyper-parameter settings are shown in the figure below. The image width and height was reduced each by a factor of 8, resulting in a total decrease of 64 times which significantly improves the computation speed. The state space for this problem is big, so the learning rate of 0.1 was used. Usually learning rate is in the order of 0.01.



```
#define INPUT_WIDTH   64 //originally 512
#define INPUT_HEIGHT  64 //originally 512
#define OPTIMIZER "RMSprop" //originally none
#define LEARNING_RATE 0.1f //originally 0.0
#define REPLAY_MEMORY 10000
#define BATCH_SIZE 128 //originally 8
#define USE_LSTM false
#define LSTM_SIZE 32
```

Fig. 1. Hyper-parameters

III. Learning strategy

The Q-Learning technique was used. Therefore rewards were needed to be assigned after every step to update the Q-Learning table. The Q-table is a NxM table where N is the number of states and M is the number of Actions possible. A state represents the current joint angles of the robot's joints. Here are the events at which the agent is rewarded positive points.

- Collide with the tube
  Rewarded 50 points and episode terminates
- Distance gets close
  Rewarded points proportional to the distance covered and episode continues

Here are the events at which the agent is rewarded negative points.

- When max episodes is reached
  Rewarded -50 points and episode terminates
- Contact ground
  Rewarded -50 points and episode terminates
- Distance increase
  Rewarded -50 points and episode continues

More on the reason why these specific rewards were chosen is in the discussion section VII. If the above were to be placed in one concise equation it would look like this

$$\begin{aligned} Reward = &- 50 * (MF + HG + M) \\ &+ 50(HT + \Delta D * 0.8 + 0.2 * \overline{\Delta D} * (1 - M)) \end{aligned}$$

Where

- MF:
  Is a binary number, set to 1 when the max frames per episode is reached
- HG:
  Is a binary number, set to 1 when the gripper touches the ground
- MA:
  Is a binary number, set to 1 when the gripper moves away from the target
- HT:
  Is a binary number, set to 1 when the gripper hits the target
- ΔD:
  Difference between the radial distance between the gripper and the target in this frame and the previous frame.
- $\overline{\Delta D}$:
  Moving average of above

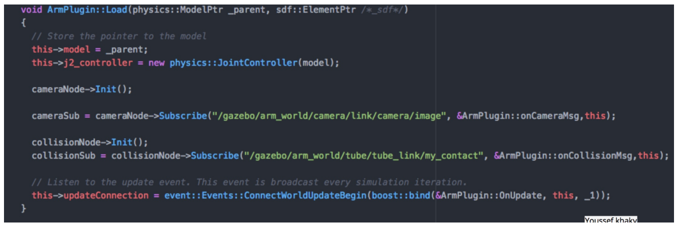
Reward awarded in this case is a hefty reward of -50 explain why not give something proportional to the distance difference?

IV. ArmPlugin.cpp

ArmPlugin is a Gazebo plugin responsible for the training of the DQN agent. In this file, the positive and negative rewards discussed earlier are set.

A. ArmPlugin::Load

This method initializes two nodes and subscribes to them. The camera node and the collision node.
The callback functions that are called upon receiving messages on these nodes are also set here.

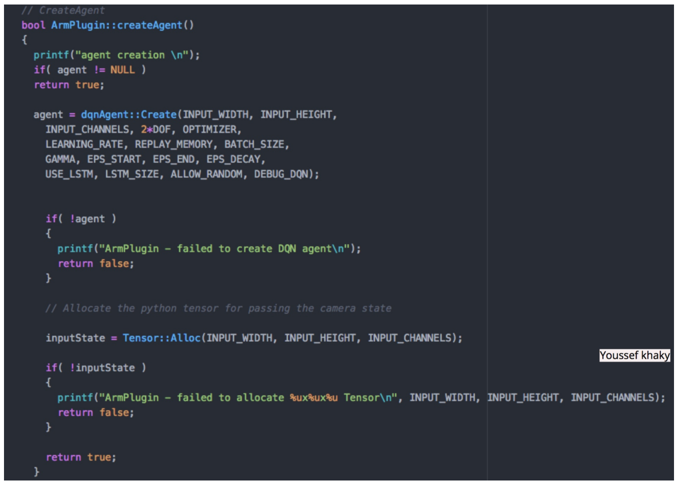


```
void ArmPlugin::Load(physics::ModelPtr _parent, sdf::ElementPtr /*_sdf*/)
{
  // Store the pointer to the model
  this->model = _parent;
  this->j2_controller = new physics::JointController(model);

  cameraNode->Init();

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

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

  // Listen to the update event. This event is broadcast every simulation iteration.
  this->updateConnection = event::Events::ConnectWorldUpdateBegin(boost::bind(&ArmPlugin::OnUpdate, this, _1));
}
```

Fig. 2. ArmPlugin::Load

B. ArmPlugin::createAgent

This method creates the DQN agent using the dqnAgent::Create method. It uses the hyper parameter discussed earlier to construct the Agent

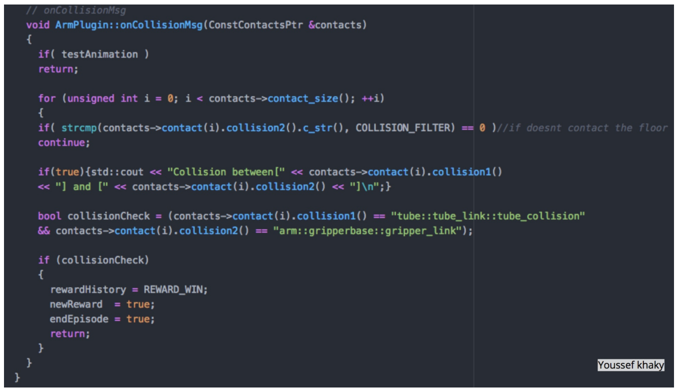


```
// CreateAgent
bool ArmPlugin::createAgent()
{
  printf("agent creation \n");
  if( agent != NULL )
  return true;

  agent = dqnAgent::Create(INPUT_WIDTH, INPUT_HEIGHT,
    INPUT_CHANNELS, 2*DOF, OPTIMIZER,
    LEARNING_RATE, REPLAY_MEMORY, BATCH_SIZE,
    GAMMA, EPS_START, EPS_END, EPS_DECAY,
    USE_LSTM, LSTM_SIZE, ALLOW_RANDOM, DEBUG_DQN);


    if( !agent )
    {
      printf("ArmPlugin - failed to create DQN agent\n");
      return false;
    }

    // Allocate the python tensor for passing the camera state

    inputState = Tensor::Alloc(INPUT_WIDTH, INPUT_HEIGHT, INPUT_CHANNELS);

    if( !inputState )
    {
      printf("ArmPlugin - failed to allocate %ux%ux%u Tensor\n", INPUT_WIDTH, INPUT_HEIGHT, INPUT_CHANNELS);
      return false;
    }

    return true;
  }
```

Fig. 3. ArmPlugin::createAgent

C. ArmPlugin::onCollisionMsg

This is a callback function that is called upon collision. Figure 4 shows that whenever collision occurs between the



```
 // onCollisionMsg
 void ArmPlugin::onCollisionMsg(ConstContactsPtr &contacts)
 {
   if( testAnimation )
   return;

   for (unsigned int i = 0; i < contacts->contact_size(); ++i)
   {
   if( strcmp(contacts->contact(i).collision2().c_str(), COLLISION_FILTER) == 0 )//if doesnt contact the floor
   continue;

   if(true){std::cout << "Collision between[" << contacts->contact(i).collision1()
   << "] and [" << contacts->contact(i).collision2() << "]\n";}

   bool collisionCheck = (contacts->contact(i).collision1() == "tube::tube_link::tube_collision"
   && contacts->contact(i).collision2() == "arm::gripperbase::gripper_link");

   if (collisionCheck)
   {
     rewardHistory = REWARD_WIN;
     newReward  = true;
     endEpisode = true;
     return;
   }
 }
}
```

Fig. 4. ArmPlugin::onCollisionMsg

target and the gripper, the collisionCheck flag gets activated causing the episode to terminate and the Agent would be awarded 50 points.

D. ArmPlugin::onCameraMsg

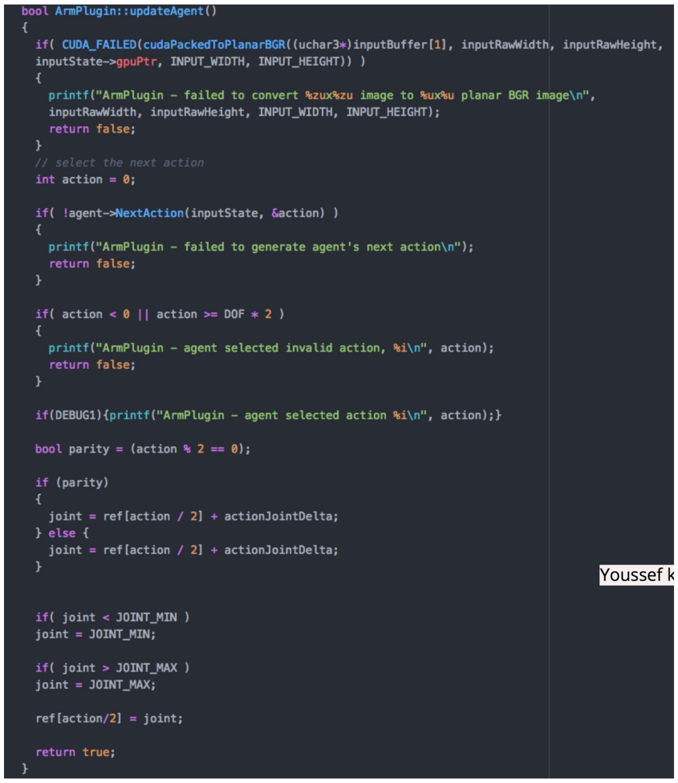
This is a callback function that is called upon receiving a new image. In that function no reward is set, it only does some miscellaneous operations such as grabbing the width and height of the image and checking the bit per pixel for the image.

E. BoxDistance

This is a utility function used to calculate the distance between two bounding boxes.

F. ArmPlugin::updateAgent

Updates the agent upon receiving a new frame

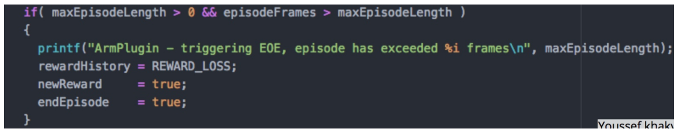


```
bool ArmPlugin::updateAgent()
{
  if( CUDA_FAILED(cudaPackedToPlanarBGR((uchar3*)inputBuffer[1], inputRawWidth, inputRawHeight,
  inputState->gpuPtr, INPUT_WIDTH, INPUT_HEIGHT)) )
  {
    printf("ArmPlugin - failed to convert %zux%zu image to %ux%u planar BGR image\n",
    inputRawWidth, inputRawHeight, INPUT_WIDTH, INPUT_HEIGHT);
    return false;
  }
  // select the next action
  int action = 0;

  if( !agent->NextAction(inputState, &action) )
  {
    printf("ArmPlugin - failed to generate agent's next action\n");
    return false;
  }

  if( action < 0 || action >= DOF * 2 )
  {
    printf("ArmPlugin - agent selected invalid action, %i\n", action);
    return false;
  }

  if(DEBUG1){printf("ArmPlugin - agent selected action %i\n", action);}

  bool parity = (action % 2 == 0);

  if (parity)
  {
    joint = ref[action / 2] + actionJointDelta;
  } else {
    joint = ref[action / 2] + actionJointDelta;
  }


  if( joint < JOINT_MIN )
  joint = JOINT_MIN;

  if( joint > JOINT_MAX )
  joint = JOINT_MAX;

  ref[action/2] = joint;

  return true;
}
```

Fig. 5. ArmPlugin::agentUpdate

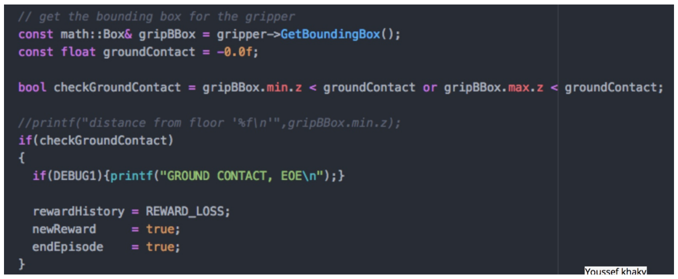
G. ArmPlugin::OnUpdate

This is an important method that has several components. Only the relevant parts of it will be addressed. Figure 6 shows one of the first things that this method checks for. If max number of frames is reached, endEpisode flag is triggered and the Agent is awarded -50 points because max number of frames were reached before the gripper or the arm was able to hit the target. If the MaxFrame is not reached, then the block of code in figure 7 is executed. This checks for weather the gripper contacts the floor. If the gripper is in contact with the floor, the eisode is terminated and agent is rewarded -50 points. If there was no contact between the gripper and the ground, the code in 8 is executed.
This block of code calculates the distance between the gripper



```
if( maxEpisodeLength > 0 && episodeFrames > maxEpisodeLength )
{
  printf("ArmPlugin - triggering EOE, episode has exceeded %i frames\n", maxEpisodeLength);
  rewardHistory = REWARD_LOSS;
  newReward     = true;
  endEpisode    = true;
}
```

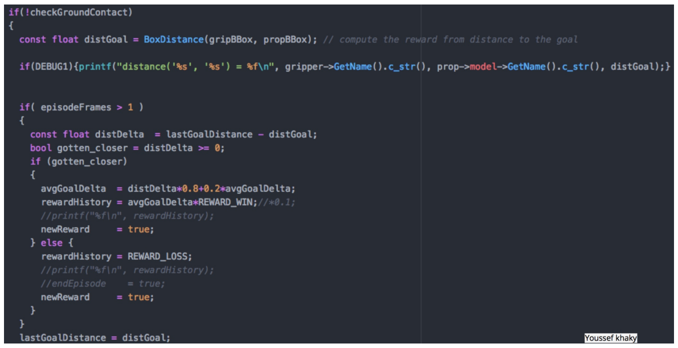
Fig. 6. ArmPlugin::onUpdate - MaxFrame termination



```
// get the bounding box for the gripper
const math::Box& gripBBox = gripper->GetBoundingBox();
const float groundContact = -0.0f;

bool checkGroundContact = gripBBox.min.z < groundContact or gripBBox.max.z < groundContact;

//printf("distance from floor '%f\n'",gripBBox.min.z);
if(checkGroundContact)
{
  if(DEBUG1){printf("GROUND CONTACT, EOE\n");}

  rewardHistory = REWARD_LOSS;
  newReward     = true;
  endEpisode    = true;
}
```

Fig. 7. ArmPlugin::onUpdate - groundContactTrue

and the target then compares the distance calculated at time t_n and t_{n-1}. If the $t_n > t_{n-1}$ this means that the gripper has gotten closer to the target and is rewarded proportionally to the distance it has gotten closer to the target by. In the event $t_n < t_{n-1}$, the gripper has gotten further and is rewarded -50 points.

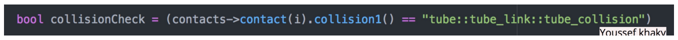


```
if(!checkGroundContact)
{
  const float distGoal = BoxDistance(gripBBox, propBBox); // compute the reward from distance to the goal

  if(DEBUG1){printf("distance('%s', '%s') = %f\n", gripper->GetName().c_str(), prop->model->GetName().c_str(), distGoal);}


  if( episodeFrames > 1 )
  {
    const float distDelta  = lastGoalDistance - distGoal;
    bool gotten_closer = distDelta >= 0;
    if (gotten_closer)
    {
      avgGoalDelta  = distDelta*0.8+0.2*avgGoalDelta;
      rewardHistory = avgGoalDelta*REWARD_WIN;//*0.1;
      //printf("%f\n", rewardHistory);
      newReward     = true;
    } else {
      rewardHistory = REWARD_LOSS;
      //printf("%f\n", rewardHistory);
      //endEpisode    = true;
      newReward     = true;
    }
  }
  lastGoalDistance = distGoal;
```

Fig. 8. ArmPlugin::onUpdate - groundContactFalse

V. Difference between implementation of two goals

There were two goals for this project. The first goal for the robotic arm to hit the target with an accuracy of 90%, TASK:A. Second one is to hit the target with the gripper part of the robotic arm with an accuracy of 80%, TASK:B. The difference in code between the two goals in the code is shown in figures 9 and 10.

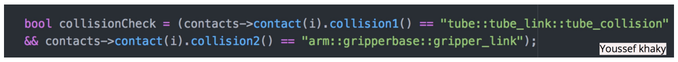


```
bool collisionCheck = (contacts->contact(i).collision1() == "tube::tube_link::tube_collision")
```

Fig. 9. Condition for Task A - Arm hits the target

VI. Results

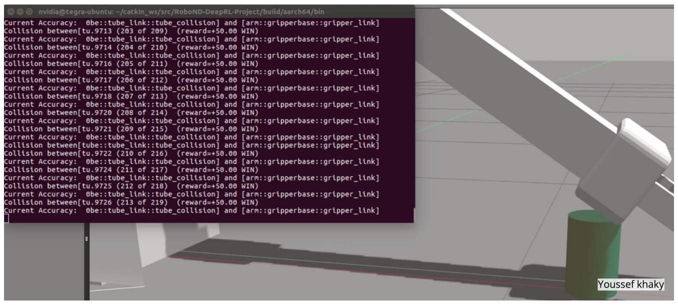
There were two goals for this project. The first goal for the robotic arm to hit the target with an accuracy of 90%. Second one is to hit the target with the gripper part of the robotic arm with an accuracy of 80%.



```
bool collisionCheck = (contacts->contact(i).collision1() == "tube::tube_link::tube_collision"
&& contacts->contact(i).collision2() == "arm::gripperbase::gripper_link");
```

Fig. 10. Condition for Task B - Gripper hits the target

Figure 11 shows the result achieved. There was no significant difference between both goals because of what is shown in figure 8. Because the robot is penalized heavily the moment the gripper gets further away from the target, the part of the robot that will always be hitting the target is always going to be the gripper regardless of the contact settings. Due to this, there is NO significant difference between both tasks and both of them generate about the same accuracy. The results differ from one run to the next because the initial weights of the DNN are different.
An accuracy around 97% was achieved in both cases for TASK A and TASK B.



```
nvidia@tegra-ubuntu: ~/catkin_ws/src/RoboND-DeepRL-Project/build/aarch64/bin
Current Accuracy:  0be::tube_link::tube_collision] and [arm::gripperbase::gripper_link]
Collision between[tu.9713 (203 of 209)  (reward=+50.00 WIN)
Current Accuracy:  0be::tube_link::tube_collision] and [arm::gripperbase::gripper_link]
Collision between[tu.9714 (204 of 210)  (reward=+50.00 WIN)
Current Accuracy:  0be::tube_link::tube_collision] and [arm::gripperbase::gripper_link]
Collision between[tu.9716 (205 of 211)  (reward=+50.00 WIN)
Current Accuracy:  0be::tube_link::tube_collision] and [arm::gripperbase::gripper_link]
Collision between[tu.9717 (206 of 212)  (reward=+50.00 WIN)
Current Accuracy:  0be::tube_link::tube_collision] and [arm::gripperbase::gripper_link]
Collision between[tu.9718 (207 of 213)  (reward=+50.00 WIN)
Current Accuracy:  0be::tube_link::tube_collision] and [arm::gripperbase::gripper_link]
Collision between[tu.9720 (208 of 214)  (reward=+50.00 WIN)
Current Accuracy:  0be::tube_link::tube_collision] and [arm::gripperbase::gripper_link]
Collision between[tu.9721 (209 of 215)  (reward=+50.00 WIN)
Current Accuracy:  0be::tube_link::tube_collision] and [arm::gripperbase::gripper_link]
Collision between[tube::tube_link::tube_collision] and [arm::gripperbase::gripper_link]
Collision between[tu.9722 (210 of 216)  (reward=+50.00 WIN)
Current Accuracy:  0be::tube_link::tube_collision] and [arm::gripperbase::gripper_link]
Collision between[tu.9724 (211 of 217)  (reward=+50.00 WIN)
Current Accuracy:  0be::tube_link::tube_collision] and [arm::gripperbase::gripper_link]
Collision between[tu.9725 (212 of 218)  (reward=+50.00 WIN)
Current Accuracy:  0be::tube_link::tube_collision] and [arm::gripperbase::gripper_link]
Collision between[tu.9726 (213 of 219)  (reward=+50.00 WIN)
Current Accuracy:  0be::tube_link::tube_collision] and [arm::gripperbase::gripper_link]
```

Fig. 11. result for both tasks

VII. Discussion

The results were much higher than expected and that is likely due to the chose of rewards, specifically the reward given when the gripper's distance from the target increases. In an earlier run, a negative reward proportional to the distance moved away from the target was given, the accuracy reached about 82% for TASK2 then it started decreasing eventually then eventually saturated to around 50%. Even though it was a negative reward, it was not deterring enough for the agent and still sometimes thought that an increase in distance in the short term can result in a more favourable outcome in the long-run. For this specific challenge, this will never be the case because because moving away from the target will never result in a favourable future and should always be avoided. Therefore the agent was very heavily penalized whenever it moved away from the target ensuring that it learns to make it an urgent priority to always get the gripper closer to the target.

VIII. Future Work

In the future these can be done:

- explore different optimizes
- include a second camera at the top

- use a robot with more DOFs such as the KUKA robot
- use LSTM
- include 2 robots equidistant from the target and make them compete