Imperial College London

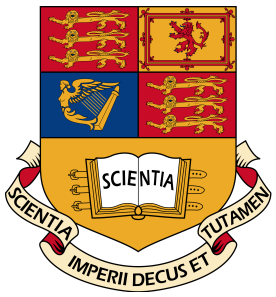
Using Deep Q Networks to train swarms of independent, identical and asynchronous agents in ROS Gazebo

SwarmbotGazebo-DQN UROP Report

Author:
Robert Holland

Supervisor:
Kai Arulkumaran

November 1, 2016



Contents

Abstract

Attempting to transfer techniques for learning Deep Q Networks trained to play Atari games to learn accurately simulated 3D robots completing asynchronous tasks. Gazebo with ROS Indigo provides a realistic robot simulator used to train networks that work in physical robots to an extent that the divide between simulation and reality will allow.

1 Introduction

Write when finished. Transition from game to realistic simulation. Transition from realistic simulation to real robots. Finish Introduction

2 Objectives

1. Configure Environment
   (a) Create world
   (b) Create creatures (swarmbots)
2. Demonstrate individual control of robots
   (a) Receive input from robot sensors via torch-ros
   (b) Create commands using Torch
   (c) Send commands to robot via torch-ros
3. Train robot to collect food
   (a) Use real time sensor information as input to a neural network that outputs commands
   (b) Optimise robot's neural network
      i. Genetic Algorithm with Fitness Proportionate Selection
      ii. Reinforcement Learning using Deep Q Networks from Kai's Atari project

3 Setup

3.1 ROS Indigo

ROS Indigo is the latest version that is compatible with the Torch-ROS library and works with ros_control.

3.2 Torch

Torch 7 is the latest version of torch..

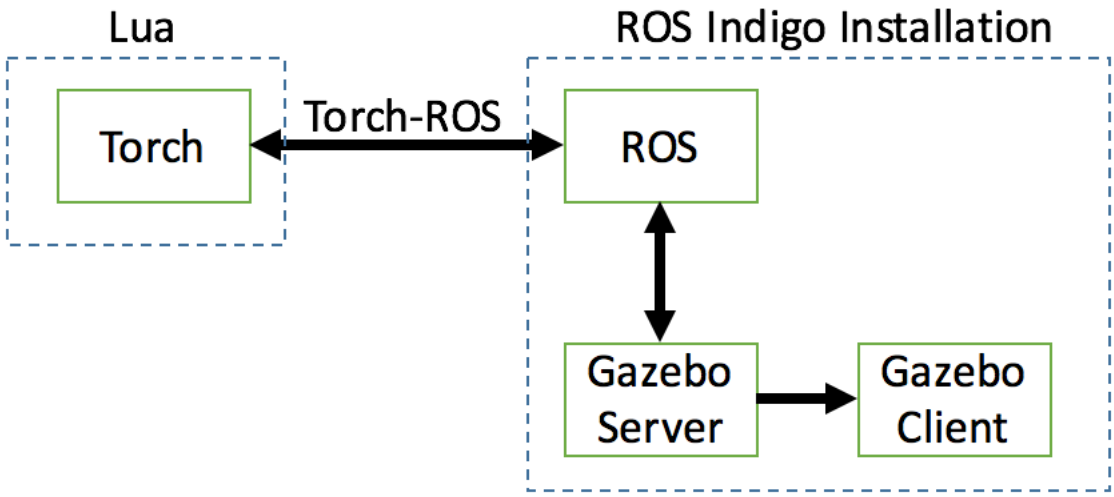


Figure 1: Code stack overview

3.3 Torch-ROS

Torch-ROS. allows Lua scripts to publish and subscribe to ROS Topics. It also can be used to set up services and service calls.

4 Code Design

4.1 Environment Design

The environment is completely encapsulated in a ROS package which can be found here.

4.1.1 Robot Design - sdf/swarm_robot_v2.sdf

ROS offers two different coding formats for robot design: SDF and URDF. While URDF is more widely used SDF is easier to learn and use which suited my relatively simple robot design.
As Gazebo is as close to real life as possible it is important to have a well built robot that offers stability and fine motor control while remaining simple and scalable. My 'swarmbot' can be built using the following tutorial.

Include figure of robot here

Gazebo offers a wide variety of sensors that can be placed on robots. I am using one RGB camera and one scan sensor, each with 90 degree field of view. Since I set both to have the same range and resolution, I can represent the state in RGBD. Plugins in the SDF file then publish the sensory input to ROS topics which I can read in Lua using Torch-ROS.
At the front of each robot is a mouth part, or bumper, which acts as a collision sensor. When this link collides with something, I can detect what type of object it is.

Include figure of robot with visualise true here

4.1.2 Virtual Environment Design - world/

In training I divided the overall task into different problems which required different worlds. I can change the environment the swarmbots inhabit by creating different world files in my ROS package.

Include figure of plus world here

Include figure of black world here

I also need to model food that the robots can detect and interact with. Food blocks can also be modeled as SDF robots, with the SDF file consisting of a single cube.

Include figure of food block here

4.2 Connecting ROS with Torch

Each of the following sections describes a script, each of which runs on independent shells on runtime. This code can be found in this repository.

4.2.1 Reward Calculation - rewards.lua

Creates tables of abstract food (food.lua) and swarmbot (swarmbot.lua) objects that hold information such as position. Each object offers services such as re-positioning the model. In particular the swarmbot object offers methods such as 'consume' and registers a callback to the collision link.
This script then creates a ROS service which takes a swarmbot ID when called, waits for that given swarmbot's collision status to be updated, and

then returns the current energy of that swarmbot. In this way the energy difference, or reward, can be calculated for a given step.
In training mode rewards.lua also ensures that food spawns around the robot. For this, the positions of the swarmbots and food need to be updated.

4.2.2 Reward Calculation - positions.lua

This script is responsible for publishing the positions of the models in simulation for other scripts. Gazebo already publishes the states of models in simulation at 1000 times per second, so this script's only function is to throttle the messages from this topic (/gazebo/model_states). This avoids a large amount of lag as updating the entire model state at 1000 times per second is expensive.

4.2.3 From Sensory Input to Command - GazeboEnv.lua

This file fits the specification provided by the rlenvs API which is used by the Atari Deep Q Learning libraries written with Torch. It is important to know that each swarmbot has it's own instance of this class, and each runs on a different thread where the thread ID corresponds to the swarmbot ID. The most important method provided by GazeboEnv.lua is 'step'. Step provides the latest action chosen by the action-value function which is then published to the command buffer. After a short duration the reward is found and returned which is used to train the network using back-propagation. The function also waits until the swarmbot sensors have been updated so that the latest observation of the environment can be returned. Waiting for the update ensures a meaningful history which can then be used to provide the next action for the subsequent call of 'step'. Finally, a terminal flag is also returned that signifies the end of the episode.

4.2.4 Command Buffer - command_buffer.lua

The swarmbots are acting asynchronously and on different threads but typical RL problems involve one agent. In the interest of keeping the problem in a familiar domain a command_buffer.lua waits to receive all commands from the GazeboEnv.lua objects and then sends all commands to swarmbots

simultaneously.

4.3 Torch

4.3.1 Network Design - SwarmbotModel.lua

SwarmbotModel.lua provides a method 'createBody' that is used by the Atari code. It returns a Torch Neural Network - I've been experimenting with simple Linear modules and CNN modules. This is the network that will be trained and validated.

5 Learning

5.1 Genetic Algorithms

Initially I attempted to learn the swarm robots using genetic algorithms. I used roulette wheel selection to allocate genes from the old generation of robots to the new generation.

5.2 Deep Q Networks and Reinforcement Learning

A more appropriate method for learning is Reinforcement Learning, especially as I am trying to learn behaviours that involve maximising the cumulative reward of an epoch.
This can be combined with Q function, or an action-value function. When given a state of the environment (for example, sensory input) this function will output an action which is then performed. After the action is complete a new environmental state is reached and the consequential reward is given. The reward can act as an error value and thus can be used to train the network with back propagation.
The creator of the code base I used to build the network explains this in more detail in this blog post.

6 Results

6.1 Genetic Algorithms

This approach was flawed for a number of reasons:

1. Population Size - Since Gazebo is an accurate simulator of real robots processing a robot is resource intensive. For Physics to work as intended the real time factor should be as close to 1.0 as possible, which constrains the number of robots to roughly 10.

   Therefore the search space of the fitness function covered by the robots was not well covered so all useful traits would have to come by mutation.

   A small population size means that the population quickly converges on an a local maximum. I used 'fitness proportionate selection' to pass on weights which meant that the median robot was identical to the best robot in a short number of generations.

2. Since real time factor needs to be close to 1.0 there is no way to speed up the simulation so that enough mutations can occur in a reasonable time

3. At the start of a new generation robots and rewards are randomly placed so the problem changes each time; this is so that the robots have good general intelligence. This means that a good trait will not necessarily make its way into the gene pool of the next generation.

4. The robots' neural networks have inputs of size 4x4x30 with the last layer holding 512 nodes to an output of size 5. Genetic algorithms are typically used for smaller networks.

6.2 Deep Q Networks and Reinforcement Learning

Figure 2: One learning robot and one validation robot with food blocks