# VirtualHome

Eneko Alaminos and Aitor Domec

September 2019

## Contents

# 1 Introduction

An autonomous agent that has to achieve a certain goal needs to know the actions that he has to take in order to complete said task. For example, maybe we want a robot to clean our room, make our bed or cook, but in order to complete these tasks he has to do some actions prior to those. In order to cook, he needs to gather the ingredients first, if he wants to clean the room he needs to take the vacuum cleaner and so on.

Knowing this, we can say that a task contains a sequence of atomic actions (sit, stand, move...) and interactions (pick-up object, turn-on object...). For example, if we want the agent to watch the TV, he will pick-up the controller, turn-on the TV and sit on the sofa.

The goal of VirtualHome is to automatically generate programs from natural language descriptions, as well as from video demonstrations, potentially allowing naive users to teach their robots a wide variety of novel tasks.

One thing to take into account is that there is a lack of databases describing activities composed of multiple steps. That is why they create a dataset about this.

To create this dataset they first crowdsource common house activities, then they formalize those activities via a Scratch interface thay they created. Finally, they implement all the atomic actions and interactions in the Unity3D game engine. Once this is done they can see how the agent behaves in the simulation environment and analyse how it behaves.

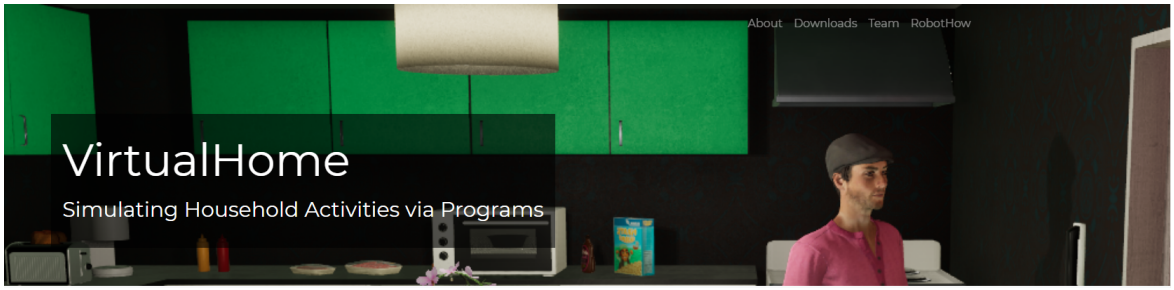


Figure 1: VirtualHome

# 2 Data Collection

The data collection has been done via crowdsourcing, the collection can be split into two key parts.

- **First**, they asked AMT workers to provide verbal descriptions of daily household activities. These activities have to be described with "high level" names such as "make coffee" or "clean the room" and describe it in detail. These activities are random activities that can happen in one of the 8 possible scenes *(living room, kitchen, dining room, bedroom, kids bedroom, bathroom, entrance hall, and home office)*. It is important to take into account that these are descriptions in 'human language'.
- **Second**, they asked the same workers to translate the descriptions into programs that would drive the robot to the fulfillment of said task via an interface built on Scratch. After this, they hired qualified workers via Upwork to double check the data.

## 2.1 Dataset Description

The programs are composed by a sequence of steps, each instruction is a Scatch block from a predefined list of 77 possible blocks. Each step in the program is defined by a block. A block defines a syntactic frame with an action and a list of arguments (e.g., the block walk requires one argument to specify the destination). To simplify the search for blocks they are organized according to 9 broad action categories, *communicate, other, body manipulation, cleaning, food, look, electronics, object manipulation, movement* and a *special block* for missing actions.
This is how a description is made:

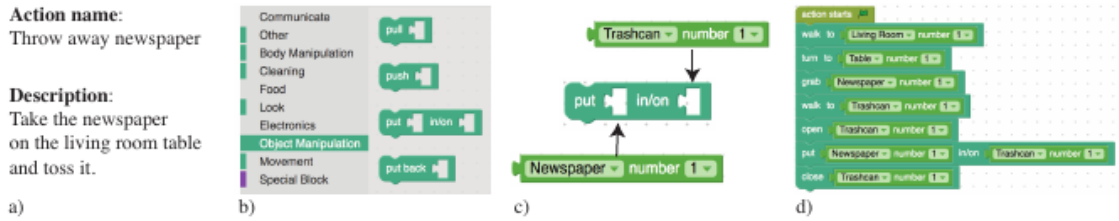


Figure 2: a) Description provided by a worker. b) User interface showing the list of block categories and 4 example blocks, c) Example of composition of a block by adding the arguments. Each block is like a Lego piece where the user can drop arguments inside and attach one block to another. d) Final program corresponding to the description from (a).

## 2.2 Dataset Analysis

They collected **1814 descriptions** from which they were able to obtain **1703 programs**. Some of those programs contained the *special block* for missing actions. They removed those programs, resulting in a total of **1257 unchecked programs**. They finally selected a set of tasks and asked workers to write programs for them, obtaining **1564 additional programs**. The resulting **2821 programs** form their ActivityPrograms dataset. On average, the collected descriptions have 3.2 sentences and 21.9 words, and the resulting programs have 11.6 steps on average. The 5 most used atomic actions are *walk, grab, find, putback* and *putobjback* and the 5 more common objects are *plate, chair, cabinet, faucet* and *bed*.

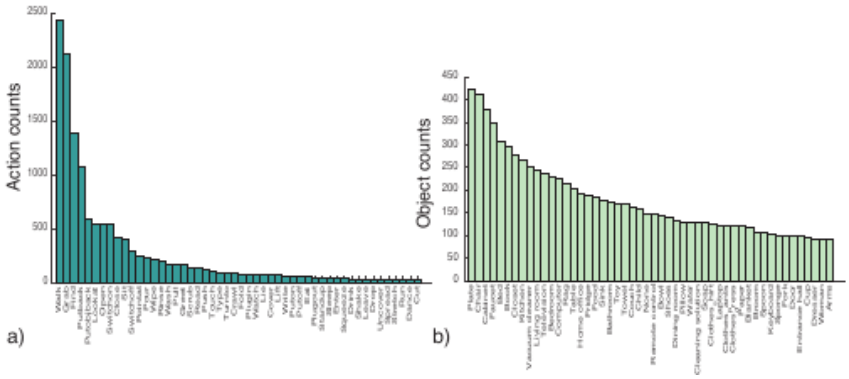


Figure 3: 50 most used objects and actions.

The dataset covers 75 atomic actions and 308 objects, making **2709 unique steps**. The diversity of the dataset is done by comparing their programs. They compute their similarities as the average length of the longest common subsequences computed between all pairs of programs. They also measure distances between activities by measuring the distance between programs. The similarity between two programs is measured as the length of their longest common subsequence of instructions divided by the length of the longest program.

Finally, to check the completeness of their programs, they asked 5 AMT workers to rate if a sample of 100 programs is complete, missing a minor step or missing a big step. Their results showed that 64% of the programs were complete, 28 % were missing minor steps and 8% were missing crucial steps.

# 3 Simulation

## 3.1 3D Environment

They have created a 3D environment where they can simulate the task with the agent. This agent will have access to all 3D and semantic information about the environment, as well as to manually defined animations. With the simulation they can record the task made by they agents to generated a video dataset.

This simulation was made in Unity3D game engine. This engine provides an *kinematic*, *physics* and *navigation* models that they can use for the simulation, as well as *3D models* provided by the users though Unity Assets store. In total there are 4 rigged humanoid models and 357 object instances for each home. For each class there are 3 differents models. For the navigation they use **Unity's NavMesh** framework, a path planner to avoid obstacles and for the animations RootMotion FinalIK inverse kinematics package.

## 3.2 Actions and animation

For every step in the program they need an animation for the corresponding action in the virtual environment. To do that they must know with object requires for doing the action and properly animate it. This is an optimization problem, were, taking in account all steps in the program, the task is to find a feasible path. For example, if the agent has to type in a keyboard, ideally the agent would type on the keyboard that is connected to the selected computer.

## 3.3 Animating Atomic actions

There are a huge variety of *atomic actions* that repeatedly appear in the database, but they have programmed the 12 most frequent ones: walk/run, grab, switch-on/off, open/close, place, look-at, sit/stand-up and touch. Although there is a large variability in how the actions are done depending on the object. Not only the agent must be animated, for example, when the agent switch-on the computer an image is shown in the screen or when a lamp is switched it must light up the room.

## 3.4 Scenes

Some 3D homes wont have objects that are mentioned in the programs, so they need to put these in the house. First, they evaluate the possibles locations for the object, then they put them before trying to execute the program.

## 3.5 Executing a Program

For the animation of the program they needed:

1. Creating a mapping between the object mentioned in the program and in the instances in the simulator.

2. For each step in the program, computing the interaction position of the agent with respect to an object.

3. Any additional information needed to animate the action, like what hand to use or the speed of the action.

To do this, they made a tree with all possibilities of assigning game objects to objects in the program, along with all interaction and attributes. To traverse the tree they use backtracking.

## 3.6 Cameras

They placed 6-9 cameras in each room of the home simulation, randomizing the position, angle and field the of view because it is important for creating a dataset with diverse video data. During the recording, a random camera that sees the agent is selected. The camera is kept until the agent is not visible anymore or when the agent makes and action and there is a camera that captures the action better than the selected one.

## 3.7 VirtualHome Activity dataset

In the dataset they collected a lot of types of interactions with different objects. The ultimate goal is to simulate this actions, but for now they support the 12 most frequent ones. So, they created another database synthesizing 5,193 programs with a simple probabilistic grammar, and then each one was described by a human annotator. Finally, they animated the programs in the simulation, generating ground-truth that allow to train and evaluate the video models. As they use a game engine, the can extract various information like segmentation, optical flow, depth texture,... as we can see in 4.

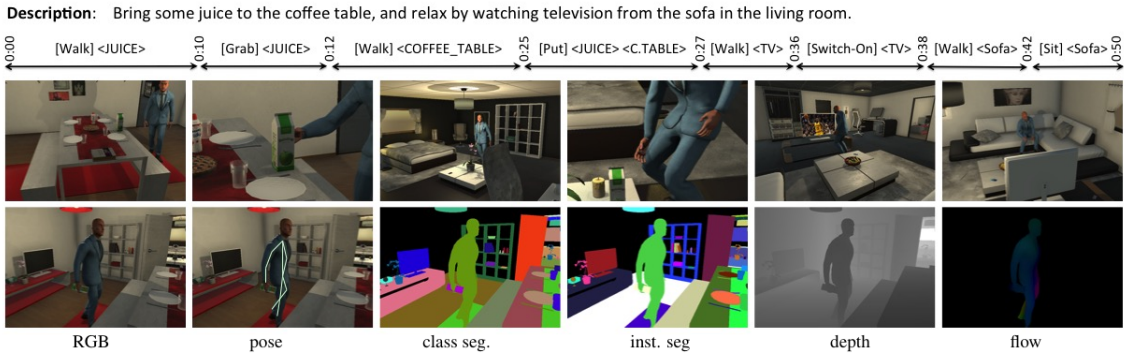


Figure 4: VirtualHome Activity Dataset

# 4 From Videos and Descriptions to Programs

The principal task that they want to cover is to generate a program for the activity from either a natural language description or a video demonstration. This task can be seen as a translation problem. To perform this task they adapted the *seq2seq model* and train it with *Reinforcement Learning*.

The model consists in two *Recurrent Neural Network's*, one encodes the input in a hidden vector representation, and the other decodes, generating one step of the program. For the encoder they use *Long Short-Term Memory*.

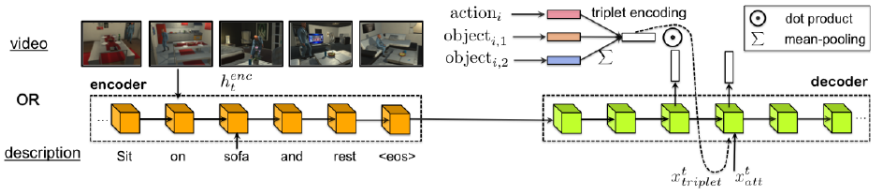


Figure 5: Encoder-decoder LSTM for generating programs from natural language descriptions or videos

## 4.1 From Description to Programs

If the input is a description, they encode each of the words using *Word2vec* and feed the result to the *LSTM* encoder and then they use a *LSTM* decoder to generate the program. This predicts an instruction at every time step by predicting his embedding. The model is trained with *cross-entropy loss* at each time step. Then they treat the program generation as an *Reinforcement Learning* problem. They use *policy gradient optimization*, a type of RL, with two rewards:

1. $r_{LCS}(w^s, g)$. For RL training, where $w^s$ is the sample program and $g$ the ground-truth one. This generate the normalized LCS metric (length of the common subsequence) between the two program to ensure that the program is semantically correct.

2. $r_{sim}(w^s)$. From the simulator. Measures whether the generated program is executable or not.

The total reward is the following: $r(w^s, g) = r_{LCS}(w^s, g) + \alpha\, r_{sim}(w^s)$

## 4.2 From Videos to Programs

To convert the videos into programs first they made a video dataset. To do that, they simulate the program in the environment choosing a random home, a random agent, adding multiple objects in the apartment and cameras as described in the previous section. It is similar like with the description's but instead of using *Word2vec* they partition each video into 2-second clips and train a model to predict the step at the middle frame. For that, they use *DilatedNet*, that gives the semantic segmentation

of each frame and with *Temporal Relation Network* predicts the embedding of an instruction. With that they can obtain the likelihood of each instruction. Then, the prediction is used as input to the *RNN* encoder for program generation.

# 5 Conclusion

They collected a large knowledge base of how-to for household activities specifically aimed for robots. The dataset contains natural language descriptions of activities that they then transform into programs that are represented as a sequence of steps. Those programs are unique because they contain all the necessary steps to perform the activity correctly. They can introduce those programs into the 3D simulator that they created (VirtualHome) in order to create a large video activity set. They also propose a model that infers a program from either video or text description, this way new robots may be driven by natural language or video demostration, at least that is the intention that they have.

They provide a Python API for communicate with the simulator. This API is easy to use as is well documented with some examples and one demo that explains the basics to use it. Although we didn't have mayor problem's using it, when it tries to generates a video, it always raises an error, so we had to made it manually with the images it generated. For now, the simulator is not perfect, it has some problems with the animations, but we think that it is a good idea to test it in a 3D virtual environment first before taking it to a real environment.

# References

[1] Xavier Puig, Kevin Ra, Marko Boben, Jiaman Li, Tingwu Wang,Sanja Fidler and Antonio Torralba. *VirtualHome: Simulating Household Activities via Programs*. In arXiv:1806.07011v1, 2018. https://arxiv.org/pdf/1806.07011.pdf

[2] VirtualHome

[3] *Upwork*, https://www.upwork.com/

[4] *Mturk*, https://www.mturk.com/

[5] seq2seq model

[6] Reinforcement learning

[7] Michael Nguyen. Illustrated guide to RNN.

[8] Recurrent Neural Network.

[9] Michael Nguyen. Illustrated guide to LSTM

[10] Long Short-Term Memory

[11] Word2vec

# 6 Appendix

## .1 seq2seq model

This is a method of encoder-decoder based machine translation, taking an input of sequence with a tag and an attention value. This can be implemented using two RNN, although it is more common to use an advanced version like LSTM.

## .2 Reinforcement Learning

It is a Machine Learning area where an agent is situated in a given environment and has to make a sequence of decisions. For each decision it receives an reward or punishment from the environment. The objective is to maximize this cumulative rewards. The environment is typically formulated as a Markov decision process.

## .3 Recurrent Neural Network

It is a class of artificial neural networks where connections between nodes form a directed graph along a temporal sequence. It is used in task like speech recognition, language translation or stock predictions. This class suffers from short-term memory, that is caused by the vanishing gradient problem. In short, this problem is created in the back propagation of the neural network. Each node calculates it's own gradient, if it is small, the previous layer will fail to learn because the gradient will get smaller in each layer. To solve this, there are two architectures, LSTM or GRU.

## .4 Long Short-Term Memory

It is an artificial recurrent neural network architecture. This has feedback connections, resolving the short-term memory problem from RNN's. It can procces single data points (like images) and entire sequences of data (speech or video). LSTM is used in task like speech recognition, speech synthesis or text generation.

## .5 Word2vec

It is a group of related models that are used to produce word embeddings. Takes a input of a large corpus of text and produces a vector space.