VirtualHome: Learning to infer programs from synthetic videos of activities in the home

by Shantanu Jain

S.B. EECS MIT, 2016

Submitted to the
Department of Electrical Engineering and Computer Science
In Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Computer Science

at the

Massachusetts Institute of Technology

September 2017

© 2017 Massachusetts Institute of Technology. All rights reserved.
VirtualHome: Learning to infer programs from synthetic videos of activities in the home

by Shantanu Jain

S.B. EECS MIT, 2016

Submitted to the
Department of Electrical Engineering and Computer Science
In Partial Fulfillment of the Requirements for the Degree of
Master of Engineering in Computer Science

Abstract
This project models complex activities that occur in a typical household. Programs – sequences of atomic actions and interactions – are used as a high-level, unambiguous representation of complex activities executable by an agent. However, no dataset of household activity programs currently exists. This project builds such a dataset by crowdsourcing programs of typical household activities, via a game-like interface used for teaching kids how to code. The most common atomic actions are implemented in the Unity3D game engine, and videos are recorded of an agent executing the collected programs in a simulated household environment. The VirtualHome simulator allows the creation of a large activity video dataset with rich ground-truth, enabling training and testing of video understanding models. Using the collected dataset, a sequence-to-sequence neural encoder-decoder model with visual attention is built, and learns to infer programs directly from videos. It achieves 46.2% accuracy for action inference.

Thesis Supervisor: Antonio Torralba
Title: Professor
Contents

1 Introduction ............................................................................................................................................ 13

1.1 Background ......................................................................................................................................... 13

1.2 Video understanding .......................................................................................................................... 14

1.3 Goals of this thesis project ................................................................................................................ 15

2 Related Work ...................................................................................................................................... 17

2.1 Representing activities as programs .................................................................................................. 17

2.2 Code generation ................................................................................................................................. 18

2.3 Robotics ........................................................................................................................................... 18

2.4 Simulators for learning algorithms ................................................................................................... 18

3 Datasets ................................................................................................................................................ 20

3.1 Dataset collection ............................................................................................................................... 20

3.1.1 Crowdsourcing activities and natural language descriptions ....................................................... 21

3.1.2 Crowdsourcing programs from activity descriptions .................................................................... 22

3.2 Dataset analysis ................................................................................................................................. 25

3.3 VirtualHome Activity – A Synthetic Dataset .................................................................................. 26

4 Inferring Programs from Descriptions .............................................................................................. 27

4.1 Learning and inference ...................................................................................................................... 28

4.2 Text-based prediction ......................................................................................................................... 29

5 Building the VirtualHome simulator .................................................................................................. 31
5.1 Animating programs in VirtualHome ................................................................. 32
  5.1.1 Animating atomic actions ............................................................................. 33
  5.1.2 Preparing the scene ...................................................................................... 33
  5.1.3 Program optimization .................................................................................. 34
  5.1.4 Animation ..................................................................................................... 34
5.2 From descriptions to animations in VirtualHome ............................................. 35
6 Inferring programs from videos generated with VirtualHome ................................ 39
  6.1 Constructing the embeddings ......................................................................... 40
  6.2 Classification performance ............................................................................. 42
  6.3 Ablation study .................................................................................................. 44
7 Conclusion ............................................................................................................. 47
8 Future Work .......................................................................................................... 49
  8.1 Implementing prepositions in the program grammar ...................................... 49
  8.2 Initial conditions .............................................................................................. 50
  8.3 Transfer learning to natural videos ................................................................. 51
9 References ............................................................................................................ 53
Acknowledgements

I would like to thank my mentor and collaborator, Xavier Puig Fernandez. He has been the major driving force behind the project, pushing it through deadlines even when faced with classwork, seemingly insurmountable bugs, and the endless dead ends every research project encounters. He has mentored me, taking my knowledge of machine learning beyond common datasets and simple models, to a level where I can work on complex real world problems. He generously took time away from his classes and internship at Google to help me learn about the project.

I extend my deepest gratitude to Professor Antonio Torralba, who has helped me to reach this stage of the project. Professor Torralba was able to grasp at the core of any problem I was having. He pushed me to work harder, raising my abilities. I thank him for his continued commitment to help me complete the project. This project would have not been possible without his help.

I extend my gratitude to Professor Sanja Fidler and her team at the University of Toronto, who contributed substantially to this project. In particular, they developed the video recording system inside the Unity Engine, along with countless other contributions.

I thank and acknowledge the rest of the Torralba Lab for their supporting role in the project.
I thank and acknowledge the Computer Science and Artificial Intelligence Laboratory (CSAIL), the Department of Electrical Engineering and Computer Science, and MIT, for providing the computing resources and the program through which I was able to conduct this project.

Finally, I thank my mother, father, and sister for their resolute support throughout the Master of Engineering program and through my whole time at the Institute. Without their support and encouragement, I would have never made it to where I stand today.
List of Figures

Figure 1. Natural Language description provided by a human annotator..........................21

Figure 2. Human annotator’s user interface, showing the selectable block categories, with four
example blocks in the “Object Manipulation” category.......................................................21

Figure 3. Example of constructing an instruction by adding its arguments. Each block is like a
Lego piece, and the user can drag-and-drop arguments into place, as well as attaching
blocks to each other in sequence.........................................................................................21

Figure 4. Final program corresponding to the description from Figure 1..............................21

Figure 5. Histogram of actions in the ActivityScripts dataset...............................................24

Figure 6. Histogram of objects in the ActivityScripts dataset..............................................24

Figure 7. Similarity matrix of activities, scored on the similarity between programs.............25

Figure 8. The model for inferring scripts from natural language descriptions. The model is a
seq2seq neural encoder-decoder model. Here, <eos> is a special token indicating end-of-
sentence. ............................................................................................................................28

Figure 9. Two examples, (a) and (b), of retrieved descriptions using the inferred programs to
compute similarity. The first program and description for each example is the ground truth,
the second program is inferred, and the second description is retrieved based on similarity
to the inferred program.........................................................................................................29

Figure 10. Four example three-dimensional model homes in VirtualHome. Notice the significant
diversity in room and object layout and appearance. On average, each home has 357
objects. Four of six homes were used for training, a fifth for validation, and all homes were used at test time.

Figure 11. The virtual agents in VirtualHome. One male and one female agent are used for training, and all agents are used at test time.

Figure 12. The VirtualHome Activity dataset contains videos of activities created with the simulator. The programs are generated from a probabilistic grammar based on the ActivityScripts dataset. Each generated program is described in natural language by a human annotator (top). Each program is animated in VirtualHome (first row, images) by randomizing the selection of home, agent, cameras, the placement of a subset of objects, the initial location of the agent, the speed of actions, and the instance of objects for interactions. Videos have ground-truth: (second row, images) time-stamp for each atomic action, 2D/3D pose, class segmentation, object instance segmentation, depth and optical flow.

Figure 13. Two example videos of the agent executing programs inferred from natural language descriptions. In the top example, note that the agent uses his left hand to open the fridge and to grab an object because his right hand already holds an item. Limitations exist; the bottom example shows the agent sitting on the toilet fully clothed.

Figure 14. Human evaluation of videos generated by VirtualHome of programs inferred from natural language descriptions. The x axis shows program inference accuracy, the y axis shows human score.

Figure 15. Architecture diagram of the video clip-to-instruction model. Model has an unrolled LSTM with visual attention. The LSTM inputs are a feature map of the frame, pose
information, and GoogleNet feature map. After mean pooling the LSTM outputs, an
instruction is recovered from the embedding.

Figure 16. PCA visualization of the instruction embeddings. Here, the instruction “PUTBACK
SPONGE CABINET” is selected, and its nearest neighbors are highlighted with orange and
yellow dots. Note how the instructions are well distributed across the space.

Figure 17. Example instruction using prepositions (proposed) to disambiguate the coffee
maker’s instance. A preposition block replaces the instance id selector, and takes an
argument of an object.
List of Tables

Table 1. Performance of model on inference; programs inferred from natural language
descriptions. Calculated on the VirtualHome Activity dataset (left) and the ActivityScripts
dataset (right). Performance is defined as the length of the longest common subsequence
between an inferred program and the ground truth program, divided by the length of the
longest script. .......................................................................................................................... 30

Table 2. Accuracy of actions, objects, and instructions inferred from two second video clips via
the video-to-instruction model. All results are obtained with ground truth segmentation,
except for the “All – DialatedNet” row, which is reported with DialatedNet segmentation. 43

Table 3. Performance of the model for the challenging task of video-to-program inference. All
results are obtained with ground truth segmentation, except for the “All – DialatedNet”
row, which is reported with DialatedNet segmentation......................................................... 43

Table 4. Accuracy of the full model and ablated models. Evaluated on the task of instruction
inference from videos................................................................................................................ 44

Table 5. Performance of the full model and ablated models. Evaluated on the task of full
program inference from videos................................................................................................ 45
1 Introduction

1.1 Background

Rapid advancements in the training and application of neural networks has led to state-of-the-art performance on computer vision tasks such as object recognition and scene recognition in still images and action recognition in videos. [1] The effective training and application of deep artificial neural networks have been enabled by the recent availability of highly parallelized GPU hardware. This hardware has increased training speeds dramatically, allowing very large networks like AlexNet, with 60 million parameters, to be trained on very large datasets like ImageNet, which has millions of images, leading to state-of-the-art performance. [2] [3]

While AlexNet was trained on the task of object recognition, deep neural networks have been applied to a variety of other tasks, with performance ranging from super-human, to worse than traditional methods -- for example, object detection (R-CNN) [4], scene recognition (Places365) [5], and semantic scene segmentation (ResNet). [6] More novel tasks such as inferring objects and scenes from sound have been addressed with models like SoundNet. [7]

One class of models that have seen widespread application is RNNs, and their closely related variant, LSTMs. These models are suited to working on sequences of data, such as sequences of characters or words, or sequences of frames from a video. This thesis project attempts to build a model which ‘understands’ a video, by training a seq2seq LSTM model on synthetic video data of an agent in a virtual home.
1.2 Video understanding

Understanding a video of an agent performing an activity requires recognizing all the actions that the agent is performing, and building a representation of how they are interconnected in order to achieve a particular goal. Autonomous agents also need to know the sequences of actions that need to be performed in order to perform an activity. An activity can be unambiguously represented as a program. A program contains a sequence of simple symbolic instructions, each referencing an atomic action (e.g. “sit”) or interaction (e.g. “pick-up object”) and a number of objects that the action refers to (e.g., “pick-up juice”). Assuming that an agent knows how to execute the atomic actions, programs provide an effective means of “driving” a robot to perform different activities. Programs can also be used as an internal representation of an activity demonstrated in a video. This project aims to infer programs from video demonstrations, potentially allowing naive users to teach their robot how to perform a wide variety of novel activities.

Towards this goal, one important missing piece is the lack of a database describing activities composed of multiple steps. This project crowdsources common-sense information about typical activities that happen in people’s homes. Scratch [8], a simple game-like interface used for teaching kids how to code, is adapted to collect programs for each of the activities together with natural language descriptions of the activities. This simple programing interface allowed us to collect 1440 programs of activities when deployed on Amazon Mechanical Turk. These programs include all instructions, even those that are considered common-sense knowledge, so that the programs can be executed unambiguously. The most common atomic actions are implemented in the Unity3D game engine (e.g. pick-up, switch on/off, sit, stand-up).
By exploiting the physics, navigation and kinematic models in the game engine, an artificial agent can execute these programs in a simulated household environment.

This paper first introduces the dataset collection effort and the program representation of complex activities in section 3. Section 4 shows a model that learns to automatically infer programs from natural language descriptions of activities. Section 5 introduces the VirtualHome simulator that allows us to create a large activity video dataset with rich ground-truth by using programs to drive an agent in a synthetic world. Finally, in section 6, the synthetic videos are used to train a pipeline of models that can infer the program directly from the video of an activity being executed by the agent. The VirtualHome simulator opens an important “playground” for both vision and robotics, allowing agents to exploit language and visual demonstration to execute novel activities in a simulated environment. The collected data and the VirtualHome simulator will be shared publicly to stimulate research in this exciting domain.

### 1.3 Goals of this thesis project

The VirtualHome project is large, with a complex data collection component, pre-processing pipeline, a large rendering component, and multiple neural network models, some assembled into a pipeline. Throughout my involvement with the project, I have contributed across all components of the project.

Specifically, I was most involved with building and tweaking the video-to-program model, as detailed in section 6. I was also heavily involved in the pre-processing pipeline, specifically implementing and modifying the embeddings, as well as extracting the GoogleNet features for all the videos, as detailed in sections 6.1 and 6.3. In the data-collection component,
I modified the interface with autocomplete when adding objects or rooms to the program. In section 8.1, I detail an extension to the program grammar with prepositions, a way to refer to object instances without referring to their instance id. In the rendering component, I was most involved in designing several different homes in the Unity3D engine for the programs to be rendered in, examples of which can be found in Figure 10.
2 Related Work

Because of this project’s large scope, there is significant prior work across many fields, including activity representation, code generation, human-robot instruction, and simulators for learning algorithms.

2.1 Representing activities as programs

Several works have represented activities as programs. In [9] the authors detect objects and actions in cooking videos and generate an “action plan” using a probabilistic grammar. These plans could be subsequently executed by the robot. In layman’s terms, the robots were able to learn to execute complex actions by simply watching videos. These authors further collected a tree bank of action plans from annotated cooking videos [10], creating a knowledge base of actions as programs for cooking. [11] attempted to translate cooking recipes into action plans using a Markov Random Field. [12] and [13] also argued for activities as a sequence of atomic steps. They aligned YouTube how-to videos with their narrations in order to parse videos into such programs. Most of these works were limited to either a small set of activities, or to a narrow domain (cooking). This project expands on this idea by creating a knowledge base about an exhaustive set of activities that humans do in their homes.

Just recently, [14] crowd-sourced scripts of people’s actions at home in the form of natural language. These were mostly comprised of one or two sentences describing a short sequence of actions. While this is valuable information, natural language is ambiguous and thus
hard to convert into a program executable on a robot. This project attempts to address inferring programs from both natural language descriptions and videos.

2.2 Code generation

There is growing interest in generating and interpreting source code. [15] Work most relevant to ours produces code given natural language inputs. [16] retrieves code snippets from StackOverflow based on language queries. Given a sentence describing conditions, [17] produces If-This-Then-That code. [18] generates a program specifying the logic of a card game given a short description of the rules. This project differs in the domain, and works with either text or video as input.

2.3 Robotics

A subfield of robotics aims to teach robots to follow instructions provided in natural language by a human tutor. However, most of the existing literature deals with a constrained problem, for example, learning to translate navigational instructions into a sequence of robotic actions. [19] [20] [21] [22] These instructions are typically lower level commands to the robot, limiting the action space, and are therefore simpler to execute. This is not the case in this project, which also considers interactions with objects, and everyday household activities which are typically far more complex.

2.4 Simulators for learning algorithms

Game engine based simulations have recently become popular to facilitate training visual models for autonomous driving [23] [24] and quadcopter flying. [25] Recently, [26]
released a simulator for target-driven indoor navigation; however it does not simulate household activities. The author is not aware of simulators at the scale of objects and actions in a home. VirtualHome is strongly inspired by the popular video game, The Sims. The Sims is a strategic video game mimicking daily household activities. Unfortunately, the source of the game is not public and thus cannot be used as part of the simulator.
3 Datasets

This project’s goal is to build a large repository of common activities that people perform in their daily lives in a home environment. These activities can include simple actions like “turn on the TV” or complex ones such as “make a coffee with milk”. This project is unique in its attempt to collect all the steps required to execute a task. Most of these steps are common sense knowledge for humans, and would not even be mentioned if one was to instruct another human. For example, in order to “watch tv,” one might describe it as “Switch on the television, and watch it from the sofa”. Here, the action “sit on the sofa” has been omitted, since it is part of the corpus of common sense knowledge. The project aims to collect exhaustive information about each activity, including any common sense steps.

Describing activities as programs has the advantage that it provides a clear and non-ambiguous description of all the steps needed to complete a task. Such programs can then be used in a multitude of applications such as instructing a robot or a virtual agent. Programs can also be used as a representation of a complex activities that involves a number of instructions, providing a way to understand and compare activities. The program representation aims to be unique for each activity.

3.1 Dataset collection

This section describes the crowdsourced dataset collection. Describing activities as programs can be a challenging task as most workers have no programing experience. The dataset collection effort is split in two simple steps – first, collecting activities and their natural
language descriptions, and second, collecting programs from these natural language descriptions.

3.1.1 Crowdsourcing activities and natural language descriptions

In the first step, workers are asked to provide names and natural language descriptions of daily activities in the home with as much detail as possible. In order to increase the diversity of the activities, the scene in which the activity should start was specified. One of eight scenes

**Activity name:**
Throw away newspaper

**Description:**
Take the newspaper on the living room table and toss it.

Figure 1. Natural Language description provided by a human annotator.

Figure 2. Human annotator’s user interface, showing the selectable block categories, with four example blocks in the “Object Manipulation” category.

Figure 3. Example of constructing an instruction by adding its arguments. Each block is like a Lego piece, and the user can drag-and-drop arguments into place, as well as attaching blocks to each other in sequence.

Figure 4. Final program corresponding to the description from Figure 1.
was selected randomly -- living room, kitchen, dining room, bedroom, kids bedroom, bathroom, entrance hall, or home office. Workers were also requested to provide a summary action name. For example, the activity name is “Read an email,” and its natural language description is “Turn on computer. Wait for it to load. Get online. Go to the email service. Open the email. Read the email.” However, natural language descriptions may be ambiguous or miss important steps that are common sense for humans. In the next section, unambiguous program representations are collected from these natural language descriptions.

3.1.2 Crowdsourcing programs from activity descriptions

In the second step, workers are shown the collected descriptions and asked to create an equivalent unambiguous representation using a custom graphical programing language. The programming interface builds on top of MIT’s Scratch project [8] designed to teach young children to write symbolic code. It was observed that workers were capable of quickly learning to produce useful programs by providing them with a carefully designed tutorial. Figure 1 - Figure 4 shows a snapshot of the programing interface.

Workers compose a program by composing instructions. Each instruction is a Scratch block from a predefined list of 69 possible blocks. These blocks were selected by analyzing the frequency of verbs and associated nouns in the collected descriptions. Each step in the program is defined by a block. A block defines a syntactic frame with an action and a variable length list of arguments. Figure 3 shows an example of the “put in/on” block with two arguments, the first to specify the object to be put, and the second to specify where the object should be put in or on. To simplify the block selection procedure, they are organized according to nine broad action categories (Figure 2). The program is required to contain all the steps, even those not explicitly
mentioned in the description, but that could be inferred from common-sense. Figure 4 shows an example of a program. Annotators were allowed to use a “special” block for missing actions, where the step can be written as free-form text. Programs using this special block will not be used in the rest of the paper, but allow for the identification of new blocks that need to be added.

More precisely, step $t$ in the program can be written as the instruction:

$$step_t = [action_t] < object_{t,1} > (id_{t,1}) ... < object_{t,n} > (id_{t,n})$$

Here, $id$ is an unique identifier (counter) of an object and helps in disambiguating different instances of objects that belong to the same class. An example of a program for the activity “watch tv” may be:

$$step_1 = [Walk] < TELEVISION > (1)$$
$$step_2 = [SwitchOn] < TELEVISION > (1)$$
$$step_3 = [Walk] < SOFA > (1)$$
$$step_4 = [Sit] < SOFA > (1)$$
$$step_5 = [Watch] < TELEVISION > (1)$$

Here, the programs defines that the television in steps 1, 2, and 5 refer to the same object instance, instance 1 of television.
Figure 5. Histogram of actions in the ActivityScripts dataset.

Figure 6. Histogram of objects in the ActivityScripts dataset.
3.2 Dataset analysis

In the first step, 1814 descriptions were collected. From those, programs for 1703 descriptions were collected. When programs with special blocks were removed, 1440 programs remained, which formed the ActivityScripts dataset. On average, the collected descriptions have 2.4 sentences and 23.6 words, and the resulting programs have 9.7 instructions on average.
The dataset covers 240 atomic actions and 68 objects. Figure 5 shows a histogram of the fifty most common actions appearing in the dataset, and Figure 6, the fifty most common objects. The similarity between activities can be measured by measuring the similarity between their respective programs. The similarity between two scripts is measured as the length of their longest common subsequence of instructions divided by the length of the longest script. Figure 7 shows the similarity matrix (sorted to better show the block diagonal structure) between different activities in the ActivityScripts dataset.

3.3 VirtualHome Activity – A Synthetic Dataset

For the experiments in the rest of the project, two datasets were created. The ActivityScripts dataset contains the 1440 scripts described in section 3.2. The VirtualHome Activity dataset is a synthetic dataset. 5,162 programs were synthesized using a simple probabilistic grammar. Each synthesized program was described in natural language by a human annotator. Although these programs were not created by human annotators, they produced reasonable activities, creating a much larger dataset of activity description-programs at a fraction of the cost in time and money.
4 Inferring Programs from Descriptions

The task of inferring a program from a natural language description is treated as a translation problem between two sequences. Using a program to represent an activity description will allow one to compare descriptions by measuring the similarity between programs, as described in section 3.2. It will also allow an agent to execute the described task. Furthermore, it allows the collection of a larger database of activities by simply collecting descriptions without requiring workers to create program representations, the most time consuming part of the workers’ tasks.

The seq2seq model [27] is adopted for this project’s translation task. This project’s model consists of an RNN-encoder that encodes the input sequence into a hidden vector representation, and another RNN acting as a decoder, generating one instruction of the program at a time. An LSTM with a 100-dimensional hidden state is the encoder. At each step $t$, the RNN-decoder decodes an instruction. Let $x_t$ denote an input vector to our RNN decoder at step $t$, and let $h^t$ be the hidden state. Here, $h^t$ is computed as in the standard LSTM using $\tanh$ as the non-linearity. Let $a_i$ be a one-hot encoding of an action $i$, and $o_i$ a one-hot encoding of an object $i$. The probability $p_i^t$ of an instruction $i$ at step $t$ is computed as:

$$\bar{a}_i = W_a a_i$$

$$\bar{o}_{i,n} = W_o o_{i,n}$$

$$v_i = mean(\bar{a}_i, \bar{o}_{i,1}, ..., \bar{o}_{i,n})$$

$$p_i^t = \text{softmax}_i(\frac{v_i}{\|v_i\|} \cdot h^t)$$
where $W_a$ and $W_o$ are learnable matrices, and $v_i$ denotes an embedding of an instruction.

The input vector $x_t$ concatenates multiple features. In particular, the embedding used is $v$, the instruction with the highest probability from the previous step. Following [27], the attention mechanism, computed over the encoder’s states, is used to get another feature $x_t^{att}$.

In particular:

$$
\alpha^t_j = (h^{t}_{enc})^T(W_{att} h^t)
$$

$$
x_t^{att} = \sum_j \alpha^t_j h^t_{enc}
$$

where $W_{att}$ are learnable parameters. $h^T_{enc}$, i.e., the last state of the encoder, is used as an additional feature. The full model is visualized in Figure 8.

![Figure 8](image)

**Figure 8.** The model for inferring scripts from natural language descriptions. The model is a seq2seq neural encoder-decoder model. Here, <eos> is a special token indicating end-of-sentence.

### 4.1 Learning and inference

To train the model, cross-entropy loss is used at each time step of the RNN-decoder. This project follows the typical training strategy where a prediction is made at each time step but feed in the ground-truth instructions to the next step. The model is trained using the Adam optimizer [28] with a batch size $b = 20$ and an initial learning rate of $\lambda = 0.001$. To exploit language priors, the word2vec [29] embeddings, trained on the Google News dataset, are used.
to initialize $W_a$ and $W_o$. Inference in the model is done by taking the instruction with the highest probability $p^t$ at time step $t$. The input, the natural language description, is represented as a one-hot encoding of each word in the description. This one-hot encoded input is fed to the RNN-encoder one step at a time.

Figure 9. Two examples, (a) and (b), of retrieved descriptions using the inferred programs to compute similarity. The first program and description for each example is the ground truth, the second program is inferred, and the second description is retrieved based on similarity to the inferred program.

### 4.2 Text-based prediction

Evaluation is done on the task of translating activity descriptions into programs. Results are reported on the ActivityScripts dataset (activity names, descriptions, and programs collected from human annotators), as well as on the VirtualHome Activity datasets. This project’s approach is compared to four baselines: 1) random sampling, where a random action and its arguments are selected for each step to construct a random program, 2) random retrieval, where a program from the training set is randomly picked, 3) skipthoughts, where the description is embedded using [30], and then the closest description from the training set in the embedding is recovered, and its program is used, and 4) a seq2seq model [22] (used to decode actions for navigation from text), which has a linear projection layer at the output of the decoder and a softmax over the instructions. Table 1 shows the results. It is observed that this project’s model outperforms all baselines on both datasets.
<table>
<thead>
<tr>
<th>Method</th>
<th>Action</th>
<th>Objects</th>
<th>Triplets</th>
<th>Mean Acc.</th>
<th>Action</th>
<th>Objects</th>
<th>Triplets</th>
<th>Mean Acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rand. Sampling</td>
<td>0.24</td>
<td>0.05</td>
<td>0.03</td>
<td>0.10</td>
<td>0.16</td>
<td>0.02</td>
<td>0.00</td>
<td>0.06</td>
</tr>
<tr>
<td>Rand. Retrieval</td>
<td>0.47</td>
<td>0.10</td>
<td>0.09</td>
<td>0.22</td>
<td>0.21</td>
<td>0.03</td>
<td>0.03</td>
<td>0.09</td>
</tr>
<tr>
<td>skipthoughts</td>
<td>0.64</td>
<td>0.31</td>
<td>0.28</td>
<td>0.41</td>
<td>0.31</td>
<td>0.19</td>
<td>0.16</td>
<td>0.19</td>
</tr>
<tr>
<td>seq2seq</td>
<td>0.67</td>
<td>0.61</td>
<td>0.56</td>
<td>0.61</td>
<td>0.32</td>
<td>0.20</td>
<td>0.16</td>
<td>0.23</td>
</tr>
<tr>
<td>Our model</td>
<td><strong>0.80</strong></td>
<td><strong>0.73</strong></td>
<td><strong>0.68</strong></td>
<td><strong>0.74</strong></td>
<td><strong>0.38</strong></td>
<td><strong>0.27</strong></td>
<td><strong>0.22</strong></td>
<td><strong>0.29</strong></td>
</tr>
</tbody>
</table>

Table 1. Performance of model on inference; programs inferred from natural language descriptions. Calculated on the VirtualHome Activity dataset (left) and the ActivityScripts dataset (right). Performance is defined as the length of the longest common subsequence between an inferred program and the ground truth program, divided by the length of the longest script.
5 Building the VirtualHome simulator

One interesting application of having program representations of activities is to use them to drive characters in a simulated 3D home environment. Simulations are useful because they allow the generation of large-scale video datasets of complex activities. A simulation can also provide ground-truth information, providing richer data than the current crowd-sourced datasets of real videos [14]. Such a dataset can be created by simply recording the agent executing programs in the simulator.

The VirtualHome simulator was implemented using the Unity3D game engine, which allows the simulator to leverage Unity3D’s kinematic, physics and navigation models, as well as user-contributed 3D models available through Unity’s Assets store. Six furnished homes and

Figure 10. Four example three-dimensional model homes in VirtualHome. Notice the significant diversity in room and object layout and appearance. On average, each home has 357 objects. Four of six homes were used for training, a fifth for validation, and all homes were used at test time.
four rigged humanoid models were obtained from the web. On average, each home contains 357 object instances (86 per room). Objects from an additional 30 object classes that appear in our collected scripts, yet are not available in Unity, were obtained via the 3D warehouse. [31] To ensure visual diversity, at least three different models per class were collected. The apartments are shown in Figure 10 and the agents in Figure 11.

![Figure 11. The virtual agents in VirtualHome. One male and one female agent are used for training, and all agents are used at test time.](image)

5.1 Animating programs in VirtualHome

Each instruction in the program requires an animation to be rendered in the simulator. First, the objects specified in the instruction, if any, needs to be matched to an object in the simulator (“game object”), and then the interaction needs to be animated. The ‘object matching’ problem is an optimization problem over all objects across all steps in the program. For example, if the program requires the agent to switch on a computer and type on a
keyboard, ideally the agent would type on the keyboard next to the chosen computer and not navigate to another keyboard attached to a different computer in a different room. This section describes the methods used to animate the agents.

5.1.1 Animating atomic actions

Figure 5 shows the huge variety and number of atomic actions that appear in the collected programs. Because of resource and time constraints only the twelve most frequent actions’ animations were implemented. Note that there is large variability in how an action is performed depending on its target object. For example, consider the “open” instruction: opening a fridge is rendered differently than opening a drawer. Unity’s NavMesh framework is used for navigation in the virtual home -- path planning, in robotics parlance. For each action, the agent’s target pose is computed, and the action is animated using RootMotion FinalIK inverse kinematics package. Some objects the agent interacts with are further animated, e.g., shaking a coffee maker, animate toast in a toaster, show a (random) photo on a computer or TV screen, light up a burner on a stove, and light up the lamps in the room, when these objects are switched on by the agent.

5.1.2 Preparing the scene

If the program refers to objects that are not already in the home, the simulator first “sets” the scene by placing all missing objects in the home, before executing the script. To place these objects in a realistic way, a database of possible object locations is collected. Human annotators are shown an object name, and they select from a list of other objects, including the
floor and the wall, where those objects are likely to be found. This is relatively rare since each home is already populated with a diverse set of objects.

5.1.3 Program optimization

To animate a program, an ‘object mapping’ between the objects in the program and the corresponding instances inside the virtual home needs to be created. For each program instruction, an ‘interaction point’ needs to be computed with respect to the agent and the object; in addition, other information (“attributes”) needed to animate the action (e.g., which hand to use or the speed of movement) is also computed.

The space of all possible assignments of virtual objects to game objects, along with all interaction positions and attributes, is represented as a tree. The tree is traversed with backtracking, and is stopped as soon as a state executing the last step is found. Since the number of possible object mappings for each step is small, and the tree pruning reduces the number of interaction positions to a few, the optimization runs in a few seconds.

5.1.4 Animation

Six to nine static cameras are placed in each room, on average twenty-six per home. During recording, cameras are randomly selected based on agent’s visibility. A randomly selected camera is used as long as the agent is visible and within allowed distance. For agent-object interaction, a camera’s perspective is adjusted to enhance the visibility of the interaction. Additional randomization on the position, angle, and field of view of each camera is applied. Randomization is important when creating a dataset to ensure diversity of the final video data.
Videos of program execution are animated as described above. Multiple videos of the same program are generated by randomizing the selection of home, agent, cameras, the placement of a subset of objects, the initial location of the agent, the speed of actions, and the instance of objects for interactions. Ground truth data is generated by the simulator, with the help of [32]. Several types of ground truth data are generated: 1) time-stamp of each instruction in the video, 2) agent’s 2D/3D pose, 3) class segmentation, 4) instance segmentation, 5) depth, 6) optical flow, 7) camera parameters. An example of the generated data is shown in Figure 12.

5.2 From descriptions to animations in VirtualHome

Figure 13 shows examples of the agent executing programs inferred from natural language descriptions. To evaluate the quality of the simulator as well as the quality of the program evaluation metrics, a human study is performed. Ten examples per level of inference accuracy were randomly selected: (a) [0.95 - 1], (b) [0.8 - 0.95], (c) [0.65 - 0.8], and (d) [0.5 - 0.65]. For each example, five human annotators watched the generated videos and judged the quality of the performed activity, given its natural language description. The results are shown in Figure 14. There is an observed positive correlation between the metrics’ scores and the human annotators’ scores. This means that the metrics are ‘in agreement’ with human evaluation and thus serve as a reasonable proxy for performance. Generally, at perfect performance the simulations received high human scores, however, there are outliers where this was not the case. This may be due to imperfect graphics. It is an indication that further improvements to the simulator are possible.
The high performance of text-based activity animation allows this project to collect data by having human annotators create synthetic videos directly via natural language, replacing the block based program synthesis used so far.

Figure 12. The VirtualHome Activity dataset contains videos of activities created with the simulator. The programs are generated from a probabilistic grammar based on the ActivityScripts dataset. Each generated program is described in natural language by a human annotator (top). Each program is animated in VirtualHome (first row, images) by randomizing the selection of home, agent, cameras, the placement of a subset of objects, the initial location of the agent, the speed of actions, and the instance of objects for interactions. Videos have ground-truth: (second row, images) time-stamp for each atomic action, 2D/3D pose, class segmentation, object instance segmentation, depth and optical flow.

Description: Get an empty glass. Take milk from refrigerator and open it. Pour milk into glass.

Description: Look at the clock then get the magazine and use the toilet. When done put the magazine on the table.

Figure 13. Two example videos of the agent executing programs inferred from natural language descriptions. In the top example, note that the agent uses his left hand to open the fridge and to grab an object because his right hand already holds an item. Limitations exist; the bottom example shows the agent sitting on the toilet fully clothed.
Figure 14. Human evaluation of videos generated by VirtualHome of programs inferred from natural language descriptions. The x axis shows program inference accuracy, the y axis shows human score.
Inferring programs from videos generated with VirtualHome

The overall goal of this project is to build and train a pipeline that can infer programs by watching videos from the VirtualHome Activity dataset. The video-to-instruction model infers individual instructions at a time, a single instruction inferred from a two second clip. These two second clips are cut from the original video, such that each clip contains a single instruction. The series of inferred instructions is turned into a program via a language model, which removes redundant instructions.

The clips are down sampled from thirty frames per second to five frames per second, to reduce redundancy in input. The two second clips are fed through DialatedNet [33] to achieve class segmentation, which is then fed through a three layer convolutional network to produce a 'segmentation feature map.' A feature map produced by the last convolutional layer of GoogleNet trained on ImageNet of the frame is computed in parallel. The agent’s pose information is generated as part of the ground truth data. All three of these pieces – the segmentation feature map, the GoogleNet feature map, and the agent pose information – are fed into an unrolled LSTM.

At each step, an attention vector is computed over the segmentation grid using the segmentation feature map and the hidden state of the LSTM, $h^t$. A linear layer takes both the LSTM output and the visual attention coefficients and outputs an embedding of an instruction.
The embeddings are mean pooled for all the time steps, which temporally smooths output. The instructions are recovered by multiplying by the transpose of the embedding matrix and taking the nearest neighboring instruction. The model is trained using cross-entropy loss. Figure 15 shows the overall video-to-instruction model.

![Architecture diagram](image)

Figure 15. Architecture diagram of the video clip-to-instruction model. Model has an unrolled LSTM with visual attention. The LSTM inputs are a feature map of the frame, pose information, and GoogleNet feature map. After mean pooling the LSTM outputs, an instruction is recovered from the embedding.

6.1 Constructing the embeddings

This section details the construction of the embeddings, which are outputted by the linear layer. It is the instructions that are embedded into a vector space, and are constructed from the Google News word2vec model. [29] Each instruction is composed of a verb and a variable number of arguments.

An instruction’s embedding is constructed as the normalized sum of the verb and any arguments’ embeddings from word2vec. In the case that a verb or argument does not have an explicit embedding in word2vec, then it is constructed. For example, the verb “PUTBACK” is not
in word2vec because it is not an English language word. So the “PUTBACK” embedding is constructed as the normalized sum of the words “put” and “back.” Any such edge cases are resolved by hand. Figure 16 shows a PCA visualization of the resulting instruction embeddings. Qualitatively, the instruction embeddings are well distributed, and the nearest neighbors of instructions are logically correct. For example, the “PUTBACK SPONGE CABINET” embedding has neighbors like “PUTBACK SPOON CABINET” and “PUTBACK SPONGE COFFEETABLE.”

Figure 16. PCA visualization of the instruction embeddings. Here, the instruction “PUTBACK SPONGE CABINET” is selected, and its nearest neighbors are highlighted with orange and yellow dots. Note how the instructions are well distributed across the space.
6.2 Classification performance

Each video is partitioned into two second clips. The video-to-instruction model is evaluated on accuracy for just the action, just the object, and the full instruction. Test-time performance is computed as the accuracy averaged across all two second clips. To test how the video-to-instruction model generalizes, the test set is further divided into videos recorded in homes seen at train time, and videos in homes unseen at train time. Table 2 shows these video-to-instruction results.

To evaluate how the quality of segmentation affects performance, two experiments were performed. The first experiment used ground truth segmentation as the input to the LSTM. The second experiment used DialatedNet for segmentation, and was performed only for all homes, as shown in Table 2. Observe that performance in unseen homes is marginally lower than in seen homes, as expected. This effect is lessened when using segmentation from DialatedNet, as opposed to ground truth segmentation. This is likely a result of the noise introduced by DialatedNet’s segmentation, which acts as a form of regularization thus reducing overfitting.

The more challenging task of video-to-program inference uses the full pipeline – the output of the language model. Evaluation of this task is more difficult; the model may infer a semantically similar program to the ground truth program, different by a few instructions. Instead of measuring accuracy, as in the video-to-instruction experiments, the similarity, as defined in section 3.2 – is reported instead. Results are reported in Table 3.
### Table 2. Accuracy of actions, objects, and instructions inferred from two second video clips via the video-to-instruction model. All results are obtained with ground truth segmentation, except for the “All – DialatedNet” row, which is reported with DialatedNet segmentation.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Action (%)</th>
<th>Object (%)</th>
<th>Instructions (%)</th>
<th>Mean Acc. (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>7.9</td>
<td>0.4</td>
<td>0.1</td>
<td>2.7</td>
</tr>
<tr>
<td>Seen homes</td>
<td>50.0</td>
<td>32.4</td>
<td>19.1</td>
<td>33.8</td>
</tr>
<tr>
<td>Unseen homes</td>
<td>31.9</td>
<td>11.7</td>
<td>8.7</td>
<td>17.4</td>
</tr>
<tr>
<td>All</td>
<td>46.2</td>
<td>31.7</td>
<td>18.7</td>
<td>32.2</td>
</tr>
<tr>
<td>All – DialatedNet</td>
<td>43.7</td>
<td>26.5</td>
<td>14.7</td>
<td>28.3</td>
</tr>
</tbody>
</table>

### Table 3. Performance of the model for the challenging task of video-to-program inference. All results are obtained with ground truth segmentation, except for the “All – DialatedNet” row, which is reported with DialatedNet segmentation.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Action</th>
<th>Object</th>
<th>Instructions</th>
<th>Mean Acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Random</td>
<td>0.47</td>
<td>0.10</td>
<td>0.09</td>
<td>0.22</td>
</tr>
<tr>
<td>Seen homes</td>
<td>0.61</td>
<td>0.35</td>
<td>0.30</td>
<td>0.42</td>
</tr>
<tr>
<td>Unseen homes</td>
<td>0.57</td>
<td>0.21</td>
<td>0.18</td>
<td>0.33</td>
</tr>
<tr>
<td>All</td>
<td>0.59</td>
<td>0.27</td>
<td>0.23</td>
<td>0.36</td>
</tr>
<tr>
<td>All – DialatedNet</td>
<td>0.54</td>
<td>0.23</td>
<td>0.20</td>
<td>0.33</td>
</tr>
</tbody>
</table>
6.3 Ablation study

To understand the relative importance of each input to the video-to-instruction model, an ablation study is performed. Three additional versions of the model were trained – each with one of the three inputs omitted. Their performance on video-to-instruction inference is reported in Table 4. Action inference performance is reduced by approximately 8% regardless of the input omitted, suggesting that action could be inferred from any of the inputs. In object inference, segmentation is the most important input (50% performance drop), and GoogleNet is the least important (2% performance drop). Surprisingly, instruction inference improved when GoogleNet was removed; the cause is not clear.

The models’ results on the more challenging task of video-to-program inference is reported in Table 5. There was much less variance in performance of the different models. Understanding the cause is an interesting area of further work.

<table>
<thead>
<tr>
<th>Model</th>
<th>Action (%)</th>
<th>Object (%)</th>
<th>Instructions (%)</th>
<th>Mean Acc. (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full model</td>
<td>46.2</td>
<td>31.7</td>
<td>18.7</td>
<td>32.2</td>
</tr>
<tr>
<td>No segmentation</td>
<td>43.4</td>
<td>19.7</td>
<td>11.4</td>
<td>24.8</td>
</tr>
<tr>
<td>No GoogleNet</td>
<td>43.0</td>
<td>31.0</td>
<td>20.6</td>
<td>31.5</td>
</tr>
<tr>
<td>No pose</td>
<td>43.6</td>
<td>27.4</td>
<td>16.5</td>
<td>29.2</td>
</tr>
</tbody>
</table>

Table 4. Accuracy of the full model and ablated models. Evaluated on the task of instruction inference from videos.
<table>
<thead>
<tr>
<th>Model</th>
<th>Action</th>
<th>Object</th>
<th>Instructions</th>
<th>Mean Acc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full model</td>
<td>0.59</td>
<td>0.27</td>
<td>0.23</td>
<td>0.36</td>
</tr>
<tr>
<td>No segmentation</td>
<td>0.58</td>
<td>0.24</td>
<td>0.20</td>
<td>0.34</td>
</tr>
<tr>
<td>No GoogleNet</td>
<td>0.59</td>
<td>0.27</td>
<td>0.24</td>
<td>0.36</td>
</tr>
<tr>
<td>No pose</td>
<td>0.54</td>
<td>0.25</td>
<td>0.21</td>
<td>0.33</td>
</tr>
</tbody>
</table>

Table 5. Performance of the full model and ablated models. Evaluated on the task of full program inference from videos.
7 Conclusion

This project collected a large dataset of household activities represented as programs. The program representation consists of a series of instructions, which consist of an action and a variable number of arguments. These programs contain all the steps, even those that are considered common sense knowledge for humans. These datasets also included natural language descriptions of the activities.

An LSTM model was built that inferred these programs from natural language descriptions alone. Performance was sufficient to replace the step of human annotators writing programs. Instead, the model was used to infer programs from human annotator provided natural language descriptions.

This project introduced the VirtualHome simulator, which implements the most common instructions from the programs of the VirtualHome Activity dataset in a 3D simulator. Videos of an agent executing the programs in the simulator were recorded. Alongside the videos, a rich set of ground-truth supplemental data, including pose information, class segmentation, instance segmentation, per-pixel depth, and optical flow, was generated per frame of the recorded videos. Synthetic videos were chosen because they can be generated easily with our framework. No natural video activity dataset exists, and creating one would be time consuming, expensive, and inflexible.

This rich and diverse video activity dataset was used to train a pipeline of models that inferred programs from videos. The video-to-instruction model watches two second clips of the
video, inferring the actions at each step. The model is an LSTM with visual attention, and each cell emits an embedding. The second model in the pipeline is the language model, which infers full programs from the sequence of instructions inferred by the video-to-instruction model. Performance is evaluated on the output of video-to-instruction and the output of the full pipeline.

Overall, the project has created a pipeline which converts natural language descriptions of common household activities into synthetic demonstration videos, which are then used to train a model which can infer the program from the video. This pipeline also makes it easy to expand the VirtualHome Activity dataset with more activities.

The long term goal of the project is to train a model which watches demonstration videos of household activities, infer programs from the video, add them to a collection of activity programs, and be able to execute the program of any activity in its collection. Towards this goal, this thesis contributes several key components – a pipeline to infer programs from synthetic videos, a simulator to generate synthetic demonstration videos, a model to infer programs from natural language descriptions of common household activities, and a dataset of activities with natural language descriptions, programs, and demonstration videos generated via the VirtualHome simulator.
8 Future Work

The work presented in this thesis is just the start of this wide-ranging project. Future work primarily focuses on implementing a data collection system and support in the simulator for more expressive forms of program instructions. This includes allowing an arbitrary number of arguments for each action, instead of just up to two, implementing prepositions, which allow relative references to the object instance, and adding initial conditions to the program. The ultimate goal is to make the program representation general enough to allow for a robot to execute any arbitrary home activity.

8.1 Implementing prepositions in the program grammar

The goal of prepositions is to identify object instances without referring to the instance id. Currently, when a human annotator is constructing a program, he builds an instruction by choosing the verb block and optionally adding arguments to the block. These arguments are references to object classes that have been added to the scene, and an instance id to disambiguate which instance of that class is being referred to. However, this makes it ambiguous and error prone for the annotators in some multiple instance cases. For example, if there are two coffee makers, one in the office, and one on the kitchen bench, the relationship between the object instances and their instance ids is unclear. If the annotator were to build an instruction to “put coffee filter instance 1 into coffee maker 1,” he is not sure whether coffee maker instance 1 refers to the one on the kitchen bench, or the one in the office.
A solution to this problem is to identify object instances with relative instance selection. Prepositions handle relative instance selection with a new category of blocks that replace the instance selector. A preposition block has an argument, which is another object block. Recursive preposition blocks are not allowed, so the object argument of the preposition block must be identified with its instance id.

In the coffee maker example, the annotator would build an instruction with prepositions to make it clear which instance is the target of the coffee filter. The instruction would instead take the form “put coffee filter instance 1 into coffee maker on top of bench instance 1.” The block is visualized in Figure 17.

![Figure 17](image.png)

Figure 17. Example instruction using prepositions (proposed) to disambiguate the coffee maker’s instance. A preposition block replaces the instance id selector, and takes an argument of an object.

Ultimately, instructions using prepositions could be ‘transpiled’ into the original program grammar, by having a ‘preprocessor’ that replaces preposition blocks by inferring instance ids once information about the home and the placement of the objects used in the program is established.

8.2 Initial conditions

Currently there is no way to specify initial conditions in the existing program grammar. As a result, the simulator implements common sense defaults for initial state. For example, appliances like a coffee maker or a stove start in the ‘off’ state, items like a sink or shower start in the ‘off’ state, and the room lights start in the ‘on’ state. Conditions on the relative initial
locations of items cannot be expressed either. An annotator cannot specify, for example, that there is a hair dryer in the bathroom; he can only specify there is a hair dryer somewhere in the home. Because the dataset attempts to include all common sense knowledge in addition to the semantically interesting actions, it is important to include these initial states.

One proposal to specify initial conditions is to take advantage of the prepositions grammar extension in section 8.1, and specify the initial conditions in the “action start flag” special box which marks the start of execution for every program. A special ‘initial state’ preposition block could be used, but only inside the “action start flag” box.

However, challenges remain. It is important that the human annotators only set the minimum required preconditions. This is because more preconditions on the scene results in fewer apartments, if any, that meet the preconditions, and therefore, fewer rendered videos for each program. Alternatives include dynamically setting up the apartments for each program’s preconditions, if possible. It seems unlikely a human annotator would do more than required, mitigating the problem.

8.3 Transfer learning to natural videos

Towards the ultimate goal of automated program inference and execution from videos, one direction of future work would be enabling the video-to-program model to work on natural videos instead of synthetic videos. Synthetic video was chosen over natural video because creating a natural video dataset would be prohibitively expensive and time consuming. However, natural videos would allow a robot to observe demonstrations carried out by humans (either live or from a pre-recorded corpus) and re-execute them.
Substituting synthetic video with natural video may not be as challenging as it first appears. Because the video-to-program model takes as input a class segmented version of the two-second clip, the graphical quality of the simulator – the textures and the quality of the 3D models – is likely less important. In layman’s terms, there is less of a visual difference in a segmented natural video and a segmented synthetic video than between an RGB natural video and an RGB synthetic video.

However, significant challenges remain. To apply transfer learning, a relatively small amount of labelled training data from the target distribution is required. Creating such a dataset – a natural equivalent of the VirtualHome Activity dataset – remains expensive and time consuming. Other challenges include the camera positions (VirtualHome had relatively limited camera positions), camera shake caused by human filming, and differences in perspective and distortion. Achieving a usable data distribution is especially difficult – while VirtualHome can execute the exact same program in multiple different homes, different humans may perform the same activity in different but equivalent ways. This remains a promising area for investigation.
9 References


[11] D. Nyga and M. Beetz, "Everything robots always wanted to know about housework (but were afraid to ask)," in IROS, 2012.


