2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
September 27 - October 1, 2021. Prague, Czech Republic

2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) | 978-1-6654-1714-3/21/$31.00 ©2021 IEEE | DOI: 10.1109/IROS51168.2021.9636342

Translating Natural Language Instructions to Computer Programs for Robot Manipulation

Sagar Gubbi Venkatesh[1,2], Raviteja Upadrashta[2], and Bharadwaj Amrutur[1,2]

***Abstract*— It is highly desirable for robots that work alongside humans to be able to understand instructions in natural language. Existing language conditioned imitation learning models directly predict the actuator commands from the image observation and the instruction text. Rather than directly predicting actuator commands, we propose translating the natural language instruction to a Python function which queries the scene by accessing the output of the object detector and controls the robot to perform the specified task. This enables the use of non-differentiable modules such as a constraint solver when computing commands to the robot. Moreover, the labels in this setup are significantly more informative computer programs that capture the intent of the expert rather than teleoperated demonstrations. We show that the proposed method performs better than training a neural network to directly predict the robot actions.**

I. INTRODUCTION

A robot that can operate alongside humans and perform a variety of tasks in unconstrained environments is a long standing vision of robotic learning. These robots need to be capable of understanding instructions in natural language from untrained users[1]. In this paper, we address the problem of programming robots using natural language.

Imitation learning has been used in recent years to learn end-to-end visuomotor policies that directly map pixels to robot actuator commands[2][3][4][5][6][7]. However, this is not the only way neural networks can be used for controlling robots. It is also possible to use sensor data such as the camera feed to construct a vector space representation of the world and then to plan a path in this space[8]. For example, an object detector can be used to find all the objects in the scene. The locations of the detected objects are used to determine the robot motion necessary to move the objects to particular positions. Although this introduces rigidity in the representation of the world, the advantages of this approach include modularity (the object detector can be replaced without modifying the rest of the system) and interpretability (the output of the object detector can be examined separately).

The majority of recent works on imitation learning have used some input device such as game controller[9], VR controller[3], visual odometry based 6-DoF position tracking using smartphones[10][11], space mouse[12], etc. to record

Fig. 1. The robot receives an instruction in natural language, say "Topple the lock", observes the scene through the camera, localizes the objects in front of it, translates the natural language instruction to a Python function block, and then executes the function.

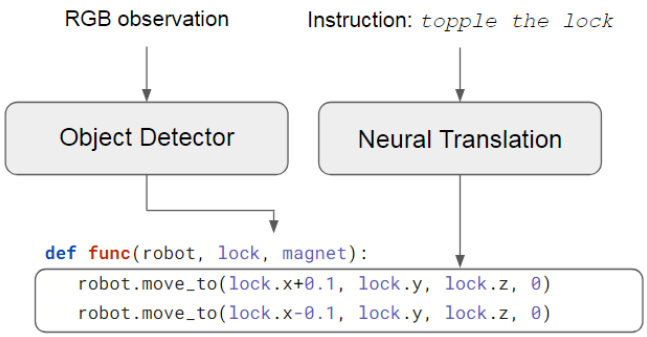

Fig. 2. The instruction in natural language is translated into a Python function block, and the output of the object detector is passed as arguments to the function.

experts teleoperating a robot. In this work, we take a different approach to collect expert demonstrations. We give a natural language instruction prompt and have experts write a Python function that controls the robot to accomplish the task specified in the instruction (Fig. 1). This function takes the output of an object detector as its argument and moves the end-effector of the robot arm to perform the specified task (Fig. 2). The dataset collected in this manner is used to train a neural network that takes a natural language instruction as input and predicts a Python function block which controls the robot when executed.

A few examples of the tasks we consider are: (a) Push the orange towards the apple, (b) Place the apple between

[1]Department of Electrical and Communication Engineering, Indian Institute of Science, Bangalore 560012, India sagar@iisc.ac.in;
[2]Robert Bosch Center for Cyber Physical Systems, Indian Institute of Science, Bangalore 560012, India ravitejaupadras@iisc.ac.in; amrutur@iisc.ac.in
**This work was supported by Robert Bosch Center for Cyber-Physical Systems

the orange and the apple, (c) Pick up the orange and use it to push the bottle off the edge of the table. Although our robot does not use a force sensor and can only move the end-effector using position control, it is possible to expand the set of primitive instructions of the robot to include complex macro instructions such as peg-in-hole insert instruction that may invoke a separately trained policy network[12]. Our approach is most suitable for "gluing" together simpler commands to compose a more complex program. A potential application for our method is in augmenting teach pendants to accept instructions in natural language.

There are several advantages of having expert demonstrations in the form of program code. One is that the expert program can invoke complex subroutines such as a constraint solver. It can be difficult to train an end-to-end neural network to copy the behavior of such complex modules. The other advantage is that the intention of the expert is clearer and less ambiguous in the program representation than in teleoperated demonstrations. For example, to "push the orange off the table", the program to perform this task clearly indicates the robot motion for different possible positions of the orange, whereas, we would need many more teleoperated demonstrations each corresponding to a different position of the object to be able to train a neural network to reliably copy the expert behavior. Finally, the program representation is more interpretable and amenable to analysis before it is executed.

Our contributions are:

- We propose an imitation learning setup where the expert demonstrations are in the form of program code and use a neural translation model to translate instructions in English to Python code that controls the robot.
- We show that the proposed method performs better than directly mapping natural language instructions to actuation commands.

The rest of this paper is organised as follows. In the following section, related work is discussed. Section 3 defines the problem statement. In Section 4, the neural network architecture that we use is described in detail. Experimental results are discussed in Section 5, and Section 6 concludes the paper.

II. RELATED WORK

Several recent papers have demonstrated that it is possible to learn visuomotor skills from human demonstrations[6][13][14][12][5][9][7]. Input devices such as VR controller[3], space mouse[12], visual odometry for 6-DoF position tracking using smartphones[10][11], etc. have been used to gather expert demonstrations. What is common to all of these approaches is that some input device is used to enable human experts to teleoperate the robot. In this work, we deviate from that approach by having experts indirectly control the robot by writing Python programs.

Understanding natural language in the context of the visual scene of the robot has been addressed by several papers. In [15], a robot system to pick and place common objects is built where the object is inferred from the input image and grounded language expressions. Understanding instructions provided in spoken language with incomplete information based on the context of the input image and common sense reasoning is addressed in [16]. The authors in [17] propose a synthetic dataset for visual question answering to debug and understand weaknesses in different grounded natural language reasoning models. In [18], the Blocks dataset is proposed to evaluate grounded spatial reasoning capabilities of neural networks. Our work also has an emphasis on spatial reasoning, but we go beyond moving a single object.

Unlike the above mentioned works, the Learning from Play (LfP) approach in [1] is goal-based imitation learning with the neural network directly controlling the actuators. Rather than conditioning on the target image, [1] replaces it with a latent vector derived from the natural language input. In this paper, we use the more traditional imitation learning approach and have experts translate natural language instructions into Python code. In Concept2Robot[19], a large dataset of human demonstrations (not teleoperated) is used to learn a reward function that is then used for training a policy network using reinforcement learning. In this work, we do not use reinforcement learning or a reward function and instead use the programs written by the expert in a fully supervised learning setting.

Much attention is devoted to object detection in the computer vision literature[20][21][22][23][24]. Although end-to-end imitation learning does not use object detection, it is also possible to use a pipelined approach where object detection is one module. For example, in [25], the pick-and-place task is performed by picking up the object at a grasp point and then bringing it near the camera for classifying to which bin the object should be placed in. In this paper, we use a fully convolutional object detector inspired by [21] to detect the positions and sizes of all the objects in the scene.

The problem of answering queries in natural language using data from a table is addressed in [26]. There are broadly two approaches to this problem. One way is to approach this as a semantic parsing problem and to generate a logical form or a SQL query from the natural language input. The other way is to process the natural language instruction along with the contents of the table to directly predict the answer. The latter approach subsumes the process of running the query into the neural network itself. In this paper, we generate Python function blocks rather than SQL statements from natural language.

In [27], the authors propose generating code from documentation strings. In [28], a pre-trained model for programming languages is proposed. A "transpiler" that translates code from one language to another is proposed in [29]. Although this paper also proposes generating program code from natural language, the end goal of controlling the robot is different. As a result, the evaluation metrics and baselines also differ. Moreover, our primary objective in this work is not to improve on code generation methods, but to show that generating code can outperform direct prediction of actuator commands.

```
def func(solver, apple, orange, banana, lemon, strawberry):
   '''place the lemon at the center, the apple just below and to the
left of lemon, the banana above the apple and on the top edge, and
the orange to the right of the banana and just below it'''
   solver.add_constraint(lemon.x == 0)
   solver.add_constraint(lemon.y == 0)

   solver.add_constraint(apple.x+apple.w/2+lemon.w/2 == lemon.x)
   solver.add_constraint(apple.y+apple.h/2+lemon.h/2 == lemon.y)

   solver.add_constraint(banana.x == apple.x)
   solver.add_constraint(banana.y+apple.h/2 == 1)

   solver.add_constraint(orange.x-orange.w/2-banana.w/2 == banana.x)
   solver.add_constraint(orange.y+orange.h/2+banana.h/2 == banana.y)
```

Fig. 3. A sample function for the arrange task that takes the width and height of all the objects and determines the positions of the objects on the table as specified by the natural language instruction (which is shown in the docstring). The Cassowary constraint solver is used to determine the positions of the objects. Note that the extents of the table on which the objects are to be placed is normalized to be in the range $[-1, 1]$.

```
def func(solver, apple, orange, banana, lemon, strawberry):
   '''place the apple at the middle of the right edge, the orange at
the right-top corner, the strawberry between the apple and orange,
and if there is enough space, the lemon between the orange and the
strawberry'''
   solver.add_constraint(apple.x+apple.w/2 == 1)
   solver.add_constraint(apple.y == 0)

   solver.add_constraint(orange.x+orange.w/2 == 1)
   solver.add_constraint(orange.y+orange.h/2 == 1)

   solver.add_constraint(strawberry.x == apple.x/2 + orange.x/2)
   solver.add_constraint(strawberry.y == apple.y/2 + orange.y/2)

   if(orange.y.value-strawberry.y.value
        -orange.h.value/2-strawberry.h.value/2 > lemon.h.value):
      solver.add_constraint(lemon.x == orange.x/2 + strawberry.x/2)
      solver.add_constraint(lemon.y == orange.y/2 + strawberry.y/2)
```

Fig. 4. A sample function for arrange task that uses the output of the constraint solver before deciding to add additional constraints.

III. TASK DESCRIPTION

We consider two different tasks where the task is specified using natural language.

A. Arrange task

This task involves taking objects from a tray and placing them at different positions on the table. The instruction in natural language along with the width and height of all the objects are the inputs and the goal is to predict the positions of the objects on the table. The motion planning to pick up the object from the tray and place it at the specified location is performed separately (this is not learned).

Figures 3 and 4 show sample programs that compute the positions of the objects for the given natural language instruction. The program uses the Cassowary constraint solver (which uses the simplex method) to declaratively specify constraints for the positions of the objects. Note that it's not entirely declarative and the program can access the intermediate solution before declaring additional constraints (Fig. 4). After the program is executed, the positions of all the objects determined by the constraint solver are used to plan the pick-and-place motion of the robot arm.

```
def func(robot, bottle):
   '''topple the bottle over the edge of the table'''
   if(bottle.x > 0.8):
       robot.move_to(bottle.x-bottle.w/2-0.1, bottle.y, bottle.z, 0)
       robot.move_to(1, bottle.y, bottle.z, 0)
   elif(bottle.x < -0.8):
       robot.move_to(bottle.x+bottle.w/2+0.1, bottle.y, bottle.z, 0)
       robot.move_to(-1, bottle.y, bottle.z, 0)
   elif(bottle.y < -0.8):
       robot.move_to(bottle.x, bottle.y+bottle.h/2+0.1, bottle.z, 0)
       robot.move_to(bottle.x, -1, bottle.z, 0)
   elif(bottle.y > 0.8):
       robot.move_to(bottle.x, bottle.y-bottle.h/2-0.1, bottle.z, 0)
       robot.move_to(bottle.x, 1, bottle.z, 0)
```

Fig. 5. A sample function for the manipulation task that takes the positions and sizes of all the objects on the table and determines the sequence of robot actions to accomplish the goal specified by the natural language instruction (which is shown in the docstring). The program can control the robot by specifying the end-effector position and the suction gripper state (on/off).

```
def func(robot, orange, strawberry):
   '''push the strawberry towards the orange'''
   theta = math.atan2(orange.y-strawberry.y,orange.x-strawberry.x)
   r = strawberry.w/2+orange.h/2
   robot.move_to(strawberry.x-r*math.cos(theta),
strawberry.y-r*math.sin(theta), strawberry.z+strawberry.d/2, theta)
   robot.move_to(orange.x-r*math.cos(theta),
orange.y-r*math.sin(theta), strawberry.z+strawberry.d/2, theta)
```

Fig. 6. A sample function for manipulation task that uses trigonometric functions in the Python standard library to compute the trajectory of the end-effector of the robot.

B. Manipulation task

This task involves manipulating objects on the table as specified by the natural language instruction. Typical tasks involve reaching for an object, pushing an object somewhere, and picking-and-placing an object. To control the robot, the action space is (a) to move the end effector of the robot to the specified position (x, y, z, r), and (b) to control the suction gripper (on/off). The robot can be controlled by emitting a sequence of end effector poses and grip commands. An object detector makes available the positions and sizes of all the objects. The goal is to take the positions and sizes of all the objects on the table and to emit a sequence of end-effector positions and gripper on/off commands.

Figures 5 and 6 show sample programs that control the robot to accomplish the task specified by the natural language instruction. Unlike the previous task, the objects are already on the table. Moreover, the program must not merely specify the desired state, but it must also directly control the robot to get to the desired state. So, the current positions of the objects are used to compute the appropriate actions.

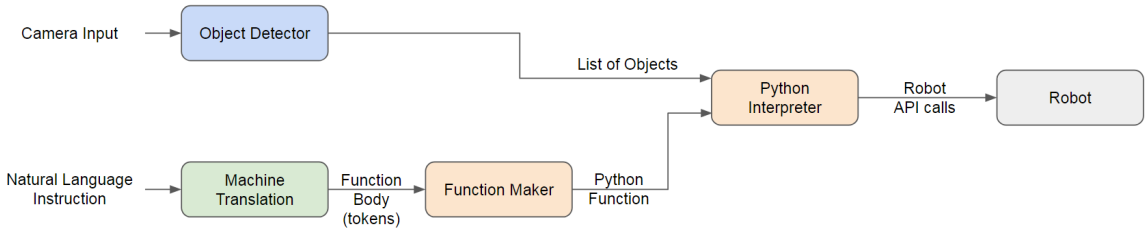

Fig. 7. Illustration of the system architecture. The natural language instruction is translated into the tokens of a Python program using neural machine translation. These tokens are assembled into an anonymous Python function. The list of objects in the scene from the object detector are passed as arguments to this function. Object classes that are not present in the scene are set to "None" when calling the generated function and will cause an exception if the generated function attempts to read their properties. When the function is executed, it communicates with the robot via a simple API consisting of "move" and "grip" commands.

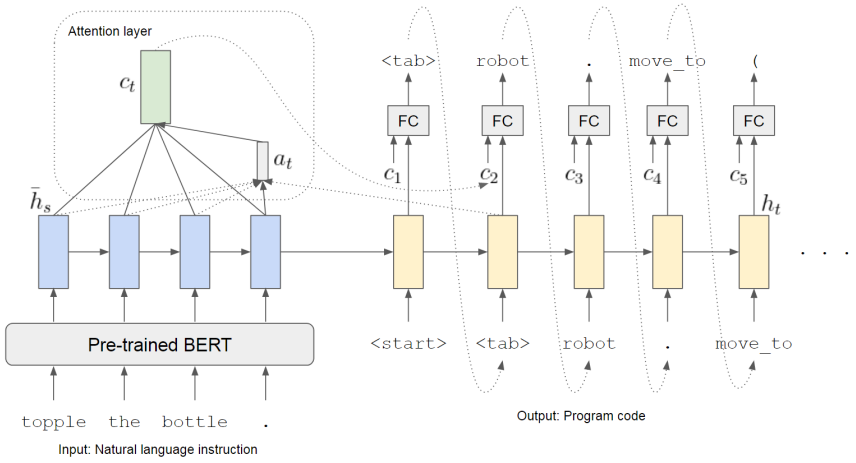

Fig. 8. The proposed neural machine translation architecture. The recurrent cells used are a single layer of LSTM cells with hidden state dimension of 1024. In the decoder, the context vector c_t is concatenated with h_t and passed through fully connected layers (FC1024-ReLU-FC100-Softmax) to predict the target sequence.

IV. NETWORK ARCHITECTURE

The overall system architecture is depicted in Fig. 7 with an example in Fig. 2. The image from the camera and the instruction in natural language are the inputs, and a Python program that drives the robot is the output. The image from the camera is processed by an object detector to obtain a list of objects in the scene. For each object in the scene, the object detector provides its position and size. We use a single shot detector[30] fine-tuned to detect the objects we use, but any object detector can be used without affecting the rest of the system. The natural language instruction is processed by a neural machine translation model described in the following paragraphs to translate the instruction into tokens of a Python function body (Fig. 8). The Python function is constructed by listing all the detected objects as arguments to the function and concatenating the generated tokens to form the function body. When executed, this function drives the robot to perform the task specified by the natural language instruction. Note that the objects that can be detected are fixed, and the translation model can only generate tokens corresponding to the known objects (present in the training set). However, the translation model can potentially generate arbitrarily complex programs.

We use an LSTM based neural machine translation model with attention[31]. Unlike most language vision models, our translation model does not take the image observation as an input. Rather, the program generated by the network accesses the attributes of the objects detected and controls the robot based on that.

The input natural language instruction is tokenized, and the embeddings for the tokens are obtained using a pre-trained BERT model[32]. Note that the BERT layers are frozen and remain unchanged during training. The input sequence embeddings are processed by an encoder LSTM with hidden states $\bar{h}_s$. After all the input tokens are processed, a decoder LSTM predicts the target sequence that is used to construct the Python function body. Our choice of using an LSTM on top of frozen BERT layers was dictated by concerns about being able to access a desktop workstation for the experiments.

At each step of the decoder, the decoder state h_t is used to attend to the input states and infer the context vector c_t that is used to predict the output y_t.

The variable length alignment vector a_t of size equal to the number of steps in the input sequence is obtained by comparing the decoder hidden state h_t with each of encoder

hidden states $\bar{h}_s$:

$$\hat{a}_t(s) = h_t^T W_a \bar{h}_s \tag{1}$$

$$a_t(s) = \frac{\exp(\hat{a}_t(s))}{\sum_{s'} \exp(\hat{a}_t(s'))} \tag{2}$$

The context vector c_t is computed as the weighted average of the hidden states of the encoder $\bar{h}_s$:

$$c_t = \sum_s a_t(s) \bar{h}_s \tag{3}$$

The context vector c_t and the decoder state h_t are concatenated and passed through fully connected layers to predict the target sequence token y_t.

V. RESULTS

There are several parts in the proposed system, and each of them can cause the robot to fail in performing the specified task. The object detector might be inaccurate, the generated program might be incorrect, or the end-effector might not successfully pick an object (for example, suction fails to lift an object). Although we are ultimately interested in whether the robot can successfully complete the task specified, it is useful to isolate the proposed machine translation model and analyze it independently. For this, we assume that the object detector is perfectly accurate and the robot is always successful in moving and picking objects. Under this ideal scenario, we first evaluate the proposed approach. Subsequently, we discuss the performance of the complete system on a real robot arm.

The proposed method is evaluated using a desktop workstation with Intel Core i7-4790K processor, 32 GB RAM, and nVidia RTX 2080Ti GPU. The models are trained using Tensorflow 1.14. A USB webcam (Logitech C310) attached to the workstation is used to capture images. The single shot object detector (from Tensorflow hub) finds the objects in the scene, and the translation model generates the program from the natural language instruction. We wrote a thin custom wrapper around the "pydobot" package to provide a simple Python API that the generated program uses to control the Dobot Magician robot arm which is connected to the workstation via USB. The pre-trained BERT model, which we use in our translation moel, is obtained from the HuggingFace Transformers library[33].

A. Datasets

1) Arrange Dataset: The arrange task involves arranging objects on the table as specified by the instruction in natural language. For this task, we have collected the arrange dataset, a parallel corpus of instructions in English and Python functions. The function takes the object sizes as arguments and sets the position of the objects as indicated in the instruction. Some examples are shown in Figs. 3 and 4. Note that in addition to the object sizes, the function is also given the Cassowary linear constraint solver[1] to specify the positions of objects as constraints to be solved. The arrange dataset has training / development / test split of 102 / 11 / 11 samples. These samples are augmented by randomly changing the object(s) simultaneously in both the instruction text and the program.

```
def func(solver, apple, orange, banana, lemon, strawberry):
    '''place the orange at the center, the strawberry below orange, apple to
the right of strawberry and on the right edge, and the lemon to the left of
the strawberry and on the left edge'''
    solver.add_constraint(orange.x == 0)
    solver.add_constraint(orange.y == 0)

    solver.add_constraint(strawberry.x == orange.x)
    solver.add_constraint(strawberry.y+strawberry.h/2+orange.h/2 == orange.y)

    solver.add_constraint(apple.x+apple.w/2 == 1)
    solver.add_constraint(apple.y == strawberry.y)

    solver.add_constraint(lemon.x+lemon.w/2 == 1)
    solver.add_constraint(lemon.y == strawberry.y)
```

Fig. 9. Sample generated program (incorrect) from the test set for the arrange task. The input instruction is in the docstring. The underlined code is incorrect. The neural network seems to have overfit for instructions with multiple phrases, and the generated code resembles a sample in the training set.

We also execute each ground truth program in the corpus for 20 different random initializations of the sizes of the objects to obtain the positions of the objects given those sizes. This secondary dataset is used for fair comparison with baseline models that directly predict the positions of the objects given the instruction and sizes of the objects.

2) Manipulation Dataset: This task involves manipulating objects already present on the table as specified by the instruction in natural language. Typical manipulation tasks in this dataset are reaching for an object, pushing an object somewhere, and picking-and-placing an object. For this task, we have collected the manipulation dataset, a parallel corpus of instructions in English and Python functions. The function takes the positions and sizes of all the objects on the table and controls the robot through an API that allows it to specify a sequence of end-effector poses and gripper states (on/off). A few examples are shown in Figs. 5 and 6. The manipulation dataset has training / development / test split of 122 / 12 / 12 samples. These are augmented by randomly changing the object(s) simultaneously in both the instruction text and the program.

For each sample in the manipulation corpus, the ground truth Python program is executed for 20 random initializations of the positions and sizes of the objects on the table and with a mock robot that records the sequence of end-effector positions and gripper state changes. This is used for fair comparison with baseline models that directly predict the sequence of end-effector poses given the instruction text and the sizes and positions of the objects.

[1]The Cassowary algorithm is used by Apple UIKit to place UI elements in GUIs

```
def func(robot, lemon):
    '''push the lemon towards the left-top'''
    theta = math.atan2(1-lemon.y,-1-lemon.x)
    r = lemon.w/2+lemon.h/2
    robot.move_to(lemon.x-r*math.cos(theta), lemon.y-r*math.sin(theta),
lemon.z+lemon.d/2, theta)
    robot.move_to(-1+lemon.w/2, 1-lemon.h/2, lemon.z+lemon.d/2, theta)
```

Fig. 10. Sample generated program (correct) from the test set for the manipulation task. The input instruction is in the docstring. The program moves the end-effector to push the object to the intended location.

```
def func(robot, apple, banana, lemon):
    '''put the apple above the banana and the lemon to the right of the banana'''
    robot.move_to(apple.x, apple.y, apple.z+apple.d/2, 0)
    robot.grip(True)
    robot.move_to(apple.x, apple.y, apple.z+apple.d/2+0.1, 0)
    robot.move_to(banana.x, banana.y+banana.h/2+apple.h/2, apple.z+apple.d/2+0.1, 0)
    robot.grip(False)

    robot.move_to(lemon.x, lemon.y, lemon.z+lemon.d/2, 0)
    robot.grip(True)
    robot.move_to(lemon.x, lemon.y, lemon.z+lemon.d/2+0.1, 0)
    robot.move_to(banana.x-banana.w/2-lemon.w/2, banana.y, lemon.z+lemon.d/2+0.1, 0)
    robot.grip(False)
```

Fig. 11. Sample generated program (incorrect) from the test set for the manipulation task. The input instruction is in the docstring. The underlined code is incorrect. The network seems to have overfit since the incorrect generated program resembles a sample in the training set.

```
def func(robot, bottle):
    '''pick up the bottle and shake it'''
    robot.move_to(bottle.x, bottle.y, bottle.z+bottle.d/2, 0)
    robot.grip(True)
    robot.move_to(bottle.x, bottle.y, bottle.z+bottle.d/2+0.1, 0)
    robot.move_to(bottle.x+0.1, bottle.y, bottle.z+bottle.d/2+0.1, 0)
    robot.move_to(bottle.x, bottle.y, bottle.z+bottle.d/2+0.1, 0)
    robot.move_to(bottle.x+0.1, bottle.y, bottle.z+bottle.d/2+0.1, 0)
    robot.move_to(bottle.x, bottle.y, bottle.z+bottle.d/2+0.1, 0)
```

Fig. 12. Sample generated program (correct) from the test set for the manipulation task. The input instruction is in the docstring. The program successfully controls the robot to perform the task.

B. Baselines

For the arrange dataset, we use LSTM+FC layers[18] as the baseline. The LSTM encodes the instruction text into a fixed size vector. This is concatenated with the sizes of all the objects and passed through several fully connected layers to directly predict the positions of all the objects.

For the manipulation dataset, we use an encoder LSTM to encode the instruction and a decoder LSTM that, at every timestep, concatenates the decoder state and the attention context vector at that timestep along with the positions and sizes of all the objects on the table, and passes this concatenated vector through fully connected layers to predict the end-effector pose and grip state[5].

C. Evaluation Metric

Evaluating a machine translation model that generates programs is challenging. Traditionally used metrics for translation models such as the BLEU score that measures the similarity between the predicted and ground truth tokens do not work well for measuring similarity of program code. The slightest change (a '+' to a '-') might give a high BLEU score but result in catastrophic failure, whereas dis-similar looking programs might actually be expressing the same logic[29]. We could simply check to see if the generated program is syntactically valid. But, this overestimates performance since many generated programs that are syntactically valid might still not accomplish the specified task. On the other hand, if we check to see if the generated program exactly matches the ground truth, we would underestimate performance since the same logic can be expressed in myriad ways. So, the best way to evaluate the generated code is to execute it and to check what it does[29]. Thus, we choose an extrinsic metric to evaluate the goodness of the generated program.

pick up strawberry and drop it past the bottom edge of table .

```
robot.move_to(strawberry.x,strawberry.y,strawberry.z+strawberry.d/2,0)
robot.grip(True)
robot.move_to(strawberry.x, ?
```

Fig. 13. Visualization of attention over the natural language instruction when predicting the token denoted by "?". When predicting the y-coordinate of the move function call, the attention layer is focusing on the "bottom" edge of the table to emit the "-1" token.

We use accuracy as the evaluation metric. Each of the predicted programs are executed 20 times with randomized object positions and sizes. Our custom robot API allows us to capture the actuator commands generated by the Python program driving the robot. During training and validation, the robot API calls (such as "move" and "grip") along with the arguments are merely recorded and not sent to the robot. Thus, we can execute both the generated program and the ground truth program and compare the resulting end-effector trajectories. For the arrange dataset, we treat the prediction to be "correct" if the absolute difference between the predicted position and ground truth position is less than 10% of the width of the table (on both x and y axes). Although the natural language instruction might admit multiple solutions (for example, "place the apple to the left of the orange" is under-specified), all the labelled data have a canonical, unambiguous target position, which eliminates any difficulty in measuring accuracy. For the manipulation dataset, the prediction is considered accurate if the absolute difference between the predicted trajectory and the ground truth trajectory is less than 10% of the width of the table at every timestep. This is merely an easy-to-evaluate proxy for whether the robot is truly accomplishing the task in the instruction. A more thorough evaluation that properly tests whether the task specified was performed successfully is conducted on a few samples with a real robot arm (Section V-E).

D. Discussion of Results

Table I compares the results of the proposed method with the baselines. All the architectures are trained using the Adam optimizer (learning rate 1e-3) for 10 epochs with batch size of 16. For both tasks, the proposed method of generating a Python program and then executing that

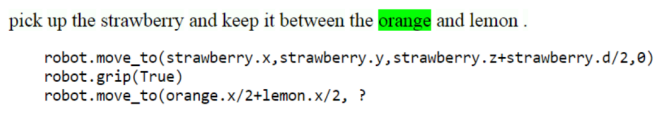

Fig. 14. Visualization of attention over the natural language instruction when predicting the token denoted by "?". When predicting the y-coordinate of the move function call, the attention layer is focusing on the "orange" in the input instruction to emit the "orange" token.

TABLE I

COMPARISON OF THE PERFORMANCE OF BASELINES (SECTION V-B) AND THE PROPOSED MODEL (SECTION IV)

Model	Arrange Task	Manipulation Task
Baseline ([18] / [5])	14.2%	9%
Proposed Seq2Seq model	80.8%	93.2%

program outperforms the baselines which directly regress the object positions (arrange dataset) or end-effector poses (manipulation dataset). The percentage of generated programs that were malformed (due to syntax errors) and could not be run were 0.6% and 2.81% in the arrange dataset and manipulation dataset respectively. The baseline models perform significantly worse on our dataset than other datasets[18] and also compare poorly with the translation model because we are attempting to train, with limited training data, a neural network to learn to solve linear constraints (Fig. 4). However, in the proposed method, the constraint solver is presented as a readily available tool which the translation model only needs to learn how to employ. In the manipulation task, the same instruction text can result in very different robot end-effector trajectories depending on the positions of the objects (for example, "push the bottle off the edge of the table"). The program captures all possible trajectories concisely and also the switch-over points when the trajectory changes because the location of the relevant object has changed (Fig. 6). In contrast, the teleoperated expert demonstrations capture the trajectory only for the position of the object in that sample and offer no clue as to when a different trajectory is suitable for the same instruction text.

Figures 9-12 show a few programs generated from the test set. Figures 13 and 14 show the attention weights for different tokens of the input instruction text when predicting a particular output token. We see that the attention mechanism is focusing on the relevant part of the instruction when predicting the program.

We have also experimented with replacing words in the instruction text with synonyms. We found that replacing "put" with "keep", "place", and "put down" always resulted in correct predictions. Likewise, we found that removing the word "the" does not change the output. Similarly, replacing "right-top corner" with only "right-top" or "top right" results in no changes to predicted sequence. However, substituting the words for objects, such as replacing "bottle" with "flask" or "pitcher" and "cup" with "chalice", caused incorrect predictions. Also, deleting the object from the instruction text resulted in incorrectly generated programs that had syntax errors. Although our datasets are small, these findings suggest that it is worth investigating the proposed method of using neural machine translation for code generation with larger datasets. Similar recent efforts using GPT-3[34] to generate code also bolster the case for further investigation.

We also found that the generalization worsens as the number of phrases in the input sequence increases (Figs. 9 and 11). There are only a few samples in the training set with 4 phrases (such as "place the orange at the bottom-right, the apple at the top-right, banana at the center, and the lemon to the right of the apple"). The model overfits on such long phrases and gives incorrect predictions that resemble the training data. However, if the input instruction is split at the commas into multiple short phrases, the model correctly predicts the positions for each of the phrases. But, this is not a viable solution because there are many instructions where such a split is not possible since the latter phrases refer to objects in the former (for example, "place the apple at the center, the orange at the top-right, and the banana in between them").

E. Demonstration on the Robot Arm

We demonstrate the complete pipeline with a Dobot Magician (Fig. 1). Common objects such as fruits, cups, magnets, etc. are used. An object detector[30] is trained to detect the position and size of these objects, but the depth (tallness from the table surface) of the object is measured beforehand and hard coded. The camera feed from an overhead camera is passed through the object detector whose output is passed as arguments to the Python function generated by the proposed method from the natural language instruction, and the function is executed. Out of 25 trials, 19 were successful with the robot accomplishing the task. All the failures were due to inaccuracies in the object detector or the suction gripper failing to pick up the object (the few longer instructions which systematically caused translation errors were not present in the small sample of instructions tested with the real robot). A video of the robot in operation is available at: `https://youtu.be/usCvsDIgWOM`

VI. CONCLUSIONS

We find that programs are rich representations of the expert demonstrations and are beneficial for learning to control robots. Moreover, the predicted programs are interpretable and easier to analyse than end-to-end neural networks that directly predict robot actions. Although this approach is necessarily constrained to those problems for which the solution can easily be expressed as a program, the proposed approach may find use in augmenting teach pendants for industrial robots to generate programs based on verbal instructions. The proposed method of generating programs is promising, so it could be worth investigating if performance can be improved by pre-training the program generator on a large corpus of source code. The proposed approach could also be useful in enabling an easily interpretable conversational system where the robot can ask clarifying questions.

ACKNOWLEDGMENT

We thank Mohammed Rizvi for his suggestions and the Robert Bosch Center for Cyber-Physical Systems for funding support.

REFERENCES

[1] C. Lynch and P. Sermanet, "Grounding language in play," *arXiv preprint arXiv:2005.07648*, 2020.

[2] S. Levine, C. Finn, T. Darrell, and P. Abbeel, "End-to-end training of deep visuomotor policies," *The Journal of Machine Learning Research*, vol. 17, no. 1, pp. 1334–1373, 2016.

[3] T. Zhang, Z. McCarthy, O. Jow, D. Lee, X. Chen, K. Goldberg, and P. Abbeel, "Deep imitation learning for complex manipulation tasks from virtual reality teleoperation," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1–8.

[4] T. Yu, C. Finn, A. Xie, S. Dasari, T. Zhang, P. Abbeel, and S. Levine, "One-shot imitation from observing humans via domain-adaptive meta-learning," *arXiv preprint arXiv:1802.01557*, 2018.

[5] R. Rahmatizadeh, P. Abolghasemi, A. Behal, and L. Bölöni, "From virtual demonstration to real-world manipulation using lstm and mdn," in *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[6] A. Giusti, J. Guzzi, D. C. Cireşan, F.-L. He, J. P. Rodríguez, F. Fontana, M. Faessler, C. Forster, J. Schmidhuber, G. Di Caro, *et al.*, "A machine learning approach to visual perception of forest trails for mobile robots," *IEEE Robotics and Automation Letters*, vol. 1, no. 2, pp. 661–667, 2015.

[7] S. G. Venkatesh, R. Upadrashta, S. Kolathaya, and B. Amrutur, "Multi-instance aware localization for end-to-end imitation learning," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020.

[8] W. Bejjani, M. R. Dogar, and M. Leonetti, "Learning physics-based manipulation in clutter: Combining image-based generalization and look-ahead planning," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019, pp. 6562–6569.

[9] R. Rahmatizadeh, P. Abolghasemi, L. Bölöni, and S. Levine, "Vision-based multi-task manipulation for inexpensive robots using end-to-end learning from demonstration," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 3758–3765.

[10] A. Mandlekar, Y. Zhu, A. Garg, J. Booher, M. Spero, A. Tung, J. Gao, J. Emmons, A. Gupta, E. Orbay, *et al.*, "Roboturk: A crowdsourcing platform for robotic skill learning through imitation," *arXiv preprint arXiv:1811.02790*, 2018.

[11] A. Mandlekar, J. Booher, M. Spero, A. Tung, A. Gupta, Y. Zhu, A. Garg, S. Savarese, and L. Fei-Fei, "Scaling robot supervision to hundreds of hours with roboturk: Robotic manipulation dataset through human reasoning and dexterity," *arXiv preprint arXiv:1911.04052*, 2019.

[12] S. Gubbi, S. Kolathaya, and B. Amrutur, "Imitation learning for high precision peg-in-hole tasks," in *2020 6th International Conference on Control, Automation and Robotics (ICCAR)*. IEEE, 2020, pp. 368–372.

[13] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[14] T. Yu, C. Finn, A. Xie, S. Dasari, T. Zhang, P. Abbeel, and S. Levine, "One-shot imitation from observing humans via domain-adaptive meta-learning," 2018.

[15] M. Shridhar and D. Hsu, "Interactive visual grounding of referring expressions for human-robot interaction," *arXiv preprint arXiv:1806.03831*, 2018.

[16] H. Chen, H. Tan, A. Kuntz, M. Bansal, and R. Alterovitz, "Enabling robots to understand incomplete natural language instructions using commonsense reasoning," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2020, pp. 1963–1969.

[17] R. Liu, C. Liu, Y. Bai, and A. L. Yuille, "Clevr-ref+: Diagnosing visual reasoning with referring expressions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2019, pp. 4185–4194.

[18] Y. Bisk, D. Yuret, and D. Marcu, "Natural language communication with robots," in *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2016, pp. 751–761.

[19] L. Shao, T. Migimatsu, Q. Zhang, K. Yang, and J. Bohg, "Concept2robot: Learning manipulation concepts from instructions and human demonstrations," in *Robotics: Science and Systems*, 2020.

[20] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[21] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[22] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," in *Advances in neural information processing systems*, 2015, pp. 91–99.

[23] S. G. Venkatesh and B. Amrutur, "One-shot object localization using learnt visual cues via siamese networks," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2019, pp. 6700–6705.

[24] S. G. Venkatesh, R. Upadrashta, S. Kolathaya, and B. Amrutur, "Teaching robots novel objects by pointing at them," in *2020 29th IEEE International Conference on Robot and Human Interactive Communication (RO-MAN)*. IEEE, 2020, pp. 1101–1106.

[25] A. Zeng, S. Song, K.-T. Yu, E. Donlon, F. R. Hogan, M. Bauza, D. Ma, O. Taylor, M. Liu, E. Romo, *et al.*, "Robotic pick-and-place of novel objects in clutter with multi-affordance grasping and cross-domain image matching," in *2018 IEEE international conference on robotics and automation (ICRA)*. IEEE, 2018, pp. 1–8.

[26] J. Herzig, P. K. Nowak, T. Müller, F. Piccinno, and J. M. Eisenschlos, "Tapas: Weakly supervised table parsing via pre-training," *arXiv preprint arXiv:2004.02349*, 2020.

[27] A. V. M. Barone and R. Sennrich, "A parallel corpus of python functions and documentation strings for automated code documentation and code generation," *arXiv preprint arXiv:1707.02275*, 2017.

[28] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, *et al.*, "Codebert: A pre-trained model for programming and natural languages," *arXiv preprint arXiv:2002.08155*, 2020.

[29] M.-A. Lachaux, B. Roziere, L. Chanussot, and G. Lample, "Unsupervised translation of programming languages," *arXiv preprint arXiv:2006.03511*, 2020.

[30] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*. Springer, 2016, pp. 21–37.

[31] M.-T. Luong, H. Pham, and C. D. Manning, "Effective approaches to attention-based neural machine translation," *arXiv preprint arXiv:1508.04025*, 2015.

[32] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[33] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, Oct. 2020, pp. 38–45. [Online]. Available: https://www.aclweb.org/anthology/2020.emnlp-demos.6

[34] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, *et al.*, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020.