Abstract

Artificial Intelligence (AI) is increasingly taking on the role of making decisions and executing actions on behalf of humans, creating a pressing need for artificial agents to truly understand human language. When agents are capable of overcoming the ambiguities of language and relate human instructions accurately to their physical world, they can be trusted to carry out the right actions that were intended. Natural language understanding (NLU) aims to enable this by going beyond recognition in natural language processing (NLP) to determine a user's intent for better human computer interaction (HCI). This paper explores the mechanics of a system that has true language understanding by combining concepts from cognitive linguistics, robotics, and NLP to model human understanding and codify human common sense. Our research illustrates that two important mechanisms are necessary for this: firstly, dissecting a natural language instruction to a language independent predicate representation; then, translating the predicate representation into grounded real-world references and constructs that an agent understands. These two mechanisms enable the agent to carry out the actions corresponding to the instruction accurately.

Index Terms - Natural Language Understanding, Human Computer Interaction, Grounded Language Learning, Cognitive Linguistics, Robotics, Natural Language Processing

1.Introduction

The problem of natural language understanding has been an enduring challenge in the development of Artificial Intelligence (AI). As AI technology becomes ubiquitous, the need for machines to understand grounded language is imperative to human-AI interaction. Winograd's (1972) SHRDLU was one of the early attempts to ground language understanding in the physical world. However, SHRDLU's requirement on laborious hard-coding of linguistic and physical rules limited its progress. Since then, there has been little progress to develop natural language understanding further. Possible reasons for this include a different focus of development in complementary fields: The robotics field's focus on mechanical optimizing; The linguistics field's focus on codifying language domains; NLP's focus on neural-network-based methodologies for better language model performance, etc. Modern devices with intelligent communication abilities may appear more competent but suffer from similar limitations of recognising more than understanding, restricting its ability to scale beyond domains the system was not trained on.

This paper builds on past work and presents a novel interdisciplinary approach for modelling true language understanding. Our approach differs from past work in that understanding takes place by dissecting complex instructions into atomic spatio-temporal concepts, followed by translation into physical groundings that are easily interpreted by machines. This is achieved through two key mechanisms: a Sentence to Predicate (STP) converter; and a Predicate to Referent (PTR) converter. We demonstrate the effectiveness of enabling a machine to truly understand language through a simulation where a machine receives complex natural

language instructions and carries out the corresponding actions accurately. This research will contribute to reconciling ambiguous natural language with the complexity of the physical world and enable progress towards human-like AI.

Our contributions are a) demonstrating the end-to-end process of parsing a natural language instruction and successfully commanding a robot to execute it, b) demonstrating the utility and limitations of cutting edge language parsers, robotics simulation software and packages, c) highlighting critical components necessary for a successful system that achieves true language understanding.

2.Related Work

Most modern NLP studies focus on text recognition and overlook the need for understanding the meaning of the text, which is why the system is unable to execute the actual intention of the user accurately. Natural language understanding (NLU) aims to address this by addressing two key limitations of recognition-based systems that power NLP today such as:

1. Context dependence: Different answers to the same question depending on context
   Question: Where are you from?
   Possible Answers: Singapore (context: birthplace)
   Singapore Management University (context: affiliations)
   Planet Earth (context: intergalactic meetings)

2. Anaphoric reference: Referencing words or phrases mentioned earlier
   Brandon went to the bank. *He* was annoyed because *it* was closed.
   A recognition-based NLP system is unable to recognise that “he” refers to Brandon and “it” refers to bank

One of the pioneering works relevant to NLU is the work of Winograd’s (1972) SHRDLU system, which involves a system programmed to understand natural language instructions containing a limited number of words and predicates. The system is able to ask questions requesting more information or proceed to execute actions corresponding to the instructions given. However, SHRDULU suffered scalability issues during the time, as the system required that all syntax and semantics of each word be hardcoded in advance, a severe constraint that limits it from scaling to handle the full complexities of language.

Since then, NLU research on language grounding has taken place across a variety of fields, primarily in robotics, computer vision and cognitive linguistics. As computer vision continues to develop, some studies also combine acquisition of knowledge in language and vision from video clips and linguistic descriptions (Alomari 2017 and Alomori 2018). Such research has pointed to the importance of cross-modal approaches for autonomous learning (Silberer and Lapata, 2012).

The idea of grounded language learning playing a role in language understanding has been researched widely. Siskind (1994) presented one of the earlier works to ground language in perception by attempting to link objects and events in stick-figure animations to language. This served as a precursor to more recent work of mapping language to actions in the video and similar multimodal approaches (Siskind, 2001; Chen and Mooney, 2008, Yu and Siskind, 2013). Previous research has suggested one way to model the human cognitive process is to define ground concepts using cognitive linguistic constructs such as spatio-temporal and action relations (Ho and Wang, 2019). The typical cognitive process will involve processing carried out on a set of ground-level atomic building blocks (Ho, 2012). Atomic operation representations can be used to ground concepts such as entities and movements in the real world and this could also allow the computer to understand what happens across time through the representation of the temporal dimension (Ho, 2013). The key difference between previous approaches and that of this paper is that we focus on learning to ground language in atomic, low-level concepts inspired by interdisciplinary fields.

Other existing approaches for explainable NLU systems tend to focus on interpreting the outputs or the connections between inputs and outputs. However, fine-grained information from inputs are often ignored (Liu, 2020). In this paper, we focus on the input sentences and attempt to achieve atomic understanding of words in a given sentence, to make sure the system truly understands the instructions given. We build on existing work on grammar parsers and the linguistic typology of the English Language to better process the input sentences ("CoreNLP", 2021 & "NLTK :: Natural Language Toolkit", 2021).

Our research aims to synthesise the best of the concepts mentioned above to enable true language understanding in machines. With an understanding of atomic representations, we could possibly reconcile complex human language instructions into interpretable atomic concepts easily understood by machines. This will be the key objective of our work.

3.Data Collection

We have built our own dictionary of atomic language concepts representing ground concepts our system will reason up from. The knowledge base is defined based on our experimentation and reiteration, while seeking inspiration from cognitive grammar definitions from Langacker (1987). The dataset is stored as a Python dictionary of base atomic language concepts and their corresponding higher level acronyms accessible by the system.

3.1 Knowledge Base

We added a good number of verb and spatial groundings based on what we have evaluated to be terms most commonly used in instructions, as shown in Tables 1, and 2. This can be further extended in the future for additional use cases.

Table 1: Examples included in verb grounding knowledge base

Verb Grounding		
No.	**Higher level word**	**Base word**
1	position	position
2	move	move
3	close	close
4	rotate	rotate
5	appear	appear
6	go	move
7	close	close
8	open	open
9	carry	lift, hold
10	hold	hold
11	lift	lift
12	pick	lift
13	pick up	lift

Table 2: Examples included in spatial grounding knowledge base

Spatial Grounding		
No.	**Higher-level word**	**Base word**
1	parallel	parallel
2	align	align
3	left	left
4	right	right
5	next to	next to
6	centre	centre
7	front	front
8	back	back
9	forward	front
10	backward	back
11	above	above
13	below	below
14	at	at
15	coincides	overlap
16	beside	beside

16	overlap	overlap
17	align	align

4.Methodology

4.1 Overall Implementation Method

The system takes in natural language sentences as input from a command line interface and passes it to the Sentence to Predicate (STP) Converter. The STP uses grammar parsers to parse sentences into standardized formats in which complex relations are broken down into atomic concepts. Predicates are then constructed and passed to the Predicate to Referent (PTR) converter. The PTR then processes the predicate and triggers a simulation of actions that corresponds to the instruction given. The two mechanisms will be elaborated further below.

4.2 Sentence to Predicate (STP) Converter

Leveraging Cutting Edge Language Parsers

The STP converter leverages a combination of outputs from the Natural Language Toolkit (NLTK) and Stanford Core NLP packages to obtain the grammatical structures of sentences. These packages are cutting edge grammar parsers, which are programs that obtain the grammatical structures of sentences by identifying, for instance, which groups of words go together (as "phrases") and which words are the subject or object of a verb. Below is an example of a sentence being parsed, with parsing results shown in Table 3.

Sentence = ‘Pick up the ball’

Table 3: Parsed text by NLTK and Stanford Core NLP

NLTK	Stanford Core NLP

`[('Pick', 'NNP'), ('up', 'RP'), ('the', 'DT'), ('ball', 'NN')]`	`(ROOT`
	`  (S`
	`  (VP (VB pick)`
	`  (PRT (RP up))`
	`  (NP (DT the) (NN ball)))))`

Linguistic Typology Inspired Implementation

Based on linguistic typology, specifically in the context of the English Language, English sentences most commonly follow the Subject-Verb-Object (SVO) sentence structure. SVO is a sentence structure where the subject comes first, the verb second, and the object third (Ramat, 2011). Below are a few examples of sentences that follow the SVO structure:

‘Can you lift the box’

‘Can you lift the box by 5 cm’

‘Tom throws the ball across the hall’

SVO languages almost always place relative clauses after the nouns which they modify and

adverbial subordinators before the clause modified. SVO languages are the second-most common order by number of known languages, after SOV languages. Table 4 below shows the 2 most common sentence structures used and some examples of other languages that follow the same structure.

Table 4: Languages following SOV and SVO sentence structure

Sentence structure	**English Equivalent**	**Proportion of languages**	**Example languages**
Subject Object Verb	“She the ball picks up”	45%	Hindi, Japanese, Korean, Latin
Subject Verb Object	“She picks up the ball”	42%	Chinese, English, French, Germen

Inspired by our linguistic typology study, we decided to construct predicates in the format of: **Verb**(Subject, Object, Goal). Once complex action relations are broken down into atomic actions, they form the **Verb** component of the predicate. We then combine the subject, object and goal state with the outputs of the grammar parser to form the complete predicate output. For complex actions, the **Goal** component is a predicate of its own because of the complexity of the goal state. If the subject and object is indicated in the parent predicate, the goal state existing as a nested predicate will inherit the subject and object from the parent predicate. Otherwise, it is automatically assumed that the subject is the robot subject initialised in the physical world.

Our STP converter is able to process instructions limited to a decent complexity which operationalises the true natural language understanding process we have conceptualised. Table 5 illustrates some examples of an instruction and its corresponding predicate output based on the spatial and verb grounding concepts it uses.

Table 5: Examples of basic instructions processed by STP Converter

Instruction	**Spatial Grounding**	**Verb Grounding**	**Predicate Output**

"move the ball to the right of the box"	to, right	move	move(, ball, to(, , right(, , of the box)))
			`s1 = "move the ball to the right of the box"`
			`print(base_form_check(get_feature_dict(s1)))`
			`Yes, this is the base form`
			`move(, ball, to(, , right(, , of the box)))`
			Referent[1]
			If not mentioned, right of the box will by default be a

[1] A default referent is allocated when the destination for the object is not explicitly stated in the instruction

			location where the object just touches the right side of the box

"move the ball to beside the box"	to, beside	move	move(, ball, to(, , beside (, , the box)))
			`s2 = "move the ball to beside the box"`
			`print(base_form_check(get_feature_dict(s2)))`
			`Yes, this is the base form`
			`move(, ball, to(, , beside(, , the box)))`

"move the ball above the box"	above	move	move(, ball, above (, , the box))
			`s3 = "move the ball above the box"`
			`print(base_form_check(get_feature_dict(s3)))`
			`Yes, this is the base form`
			`move(, ball, above(, , the box))`

"align the claw to the ball"	to	align	align(, claw, to(, , the ball))
			`s4 = "align the claw to the ball"`
			`print(base_form_check(get_feature_dict(s4)))`
			`Yes, this is the base form`
			`align(, claw, to(, , the ball))`

"lift the ball"	-	lift	lift(,ball,)
			`s5 = "lift the ball"`
			`print(base_form_check(get_feature_dict(s5)))`
			`Yes, this is the base form`
			`lift(, ball, )`

Commonly in language, sentences may appear different but have the same meaning. We have tackled this problem and made sure that as long as the sentences have the same grounded meaning, the STP will produce the same predicate output even for sentences with different syntactic structures. Table 6 provides some examples of such sentences that have the same meaning but different syntactic structures.

Table 6: Examples of sentences that have same meaning but different syntactic structures

Instruction	**Spatial Grounding**	**Complex verb vs atomic Grounded verb**	**Predicate Output**
Pick up the ball Pick the ball up	-	Pick up	Pick up (, the ball, up(, ,))
	up	Pick	Pick (, the ball, up(, ,))
Put down the ball Put the ball down	-	Put down	Put down (, the ball, down (, ,))
	down	Put	Put (, the ball, down (, ,))

4.3 Predicate to Referent (PTR) Converter

4.3.1 Key Features of PTR

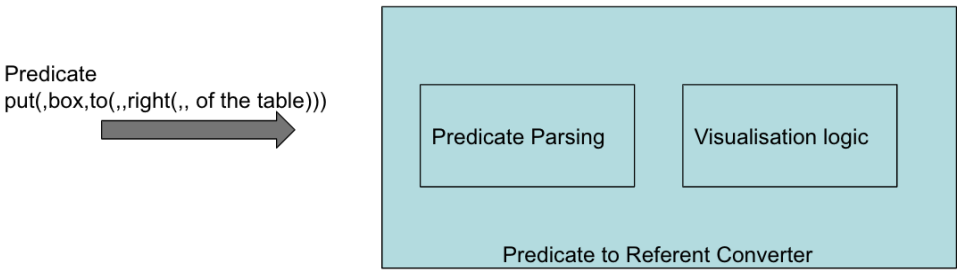


Figure 1. Key Features of the PTR

Through our experimentation and development of the PTR (Figure 1), we made a key discovery that the PTR requires two key features. Firstly, the predicate parsing process needs to be standardised and generalisable. Previously, we had assumed that parsing the input into the PTR would be trivial, but we learnt that this is a critical component of the functionality of the PTR, and had to reconsider the standardised and generalisable format of the predicate input when building the STP. We eventually standardised this format as stated in the previous section after multiple iterations, which works well for our use case.

Secondly, the visualisation software of choice that will connect to the PTR has to allow us to write custom functions that can be called by external python programs for our NLU system to integrate seamlessly to the simulation world. Previously, we assumed that any software with a Python API would be trivial to integrate with our modular system, but faced immense difficulty integrating our PTR with simulation software that does not have a Python API. Having a simulation environment with a built-in physics engine will be critical for 3D visualisation as hardcoding the effect of gravity is not scalable. This is why we eventually settled for a 2D simulation environment using Stanford Karel, while learning to use the Robotic Operating System (ROS) in parallel. Due to the time constraints of the project, we could only integrate our system with Stanford Karel because of the steeper learning curve in learning ROS. However, we built a pick-and-place sequence using ROS, as shown in appendix A, as a proof of concept for when we can successfully integrate it into our NLU system. We experimented but eventually rejected the use of AutoCAD, Jupyter-ROS and Python string method, for reasons further elaborated in appendix B.

4.3.2 PTR Mechanics: Codifying Human Common Sense and Action

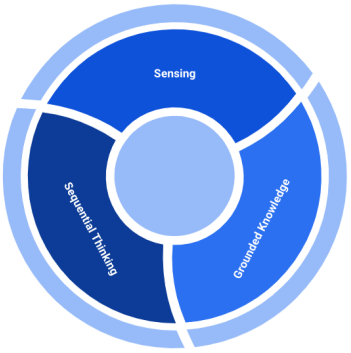


Figure 2: Three key elements of understanding

In order for a machine to understand an input predicate and relate it to its physical world, it requires three key elements of understanding: sensing, grounded knowledge and sequential thinking. Sensing refers to the system's ability to detect the current state of the world, and update this awareness of state based on changes that occur. This can be done through object detection mechanisms in 3D settings, or exist as known x-y coordinates in a 2D setting. The location of each object relative to the agent should be available to the agent at any point of time. This should ideally exist as a global state accessible and mutable by any change in the environment, in order for the agent to have the most updated state at all times.

The second key element is grounded knowledge, which refers to known spatio-temporal constructs relative to the physical environment, as well as the knowledge of actions the system is capable of executing and how they alter the state of the physical environment. The spatio-temporal constructs are defined based on the construct of the agent's environment, and all actions should be capable of mutating the global state of the physical environment.

The last component is sequential thinking, which refers to the system's ability to "think" linearly in a logical manner. This process has to be standardised and generalisable to tasks of different variation. The overarching approach that we developed to codify this linear thought process is a three-step sequence: firstly, problem decomposition, which involves parsing the predicate input into its constituent parts and allocating them to their corresponding processes; then making sense of the constituent parts by processing them based on grounded knowledge of known spatio-temporal concepts and actions. These are the atomic concepts and groundings defined in our knowledge base; finally, executing the corresponding action based on the system's understanding from consolidating the constituent parts with the system's grounded knowledge. We discovered through multiple iterations that this process works and generalises well for a wide variety of tasks.

4.4 Full System Integration

With the aforementioned components and basic constructs, we now describe the end-to-end process for a prototypical case of how the system will process complex natural language instructions and command a robot.

An instruction will be issued to the NLU system via the command line, such as “put the diamond on the green table”. Through the STP transformation process described above, this is translated into the following predicate representation:

put(robot(*inferred*), diamond, **on**(robot*(inferred)*, diamond*(inferred)*, the green table))

To carry out this action, the PTR process discussed above is engaged to derive a concrete goal configuration of **on**(robot*(inferred)*, diamond*(inferred)*, the green table) based on the NLU system’s grounded knowledge. Next, a problem solving process is triggered to derive the necessary actions required by the robot to deliver a diamond to the designated destination. There could be more than one possible solution for the problem solving process, but the robot prioritises the simplest one. A visualisation of this complete process will be elaborated in the experiment section below.

5.Experiments

5.1 The 2D Simulation Environment

To conduct our true language understanding experiments we integrated a 2D simulation world. In this world (Figure 3), a single robot agent, Karel, located at the x-y coordinates of (1,1), contains a virtual “bag” of diamonds that it can place in any location in the world. Karel perceives its world via a spatio-temporal awareness of the world in x and y coordinates, as well as receiving natural language instructions via the command line, which is the natural language interface that humans will interact with to command Karel to do tasks.

One can specify the general configuration of the layouts and possible objects in this environment. Below (Figure 3) is an example of a simulation environment we set up for the demonstration and experiments. The world contains a red box, a green table and a blue tower.

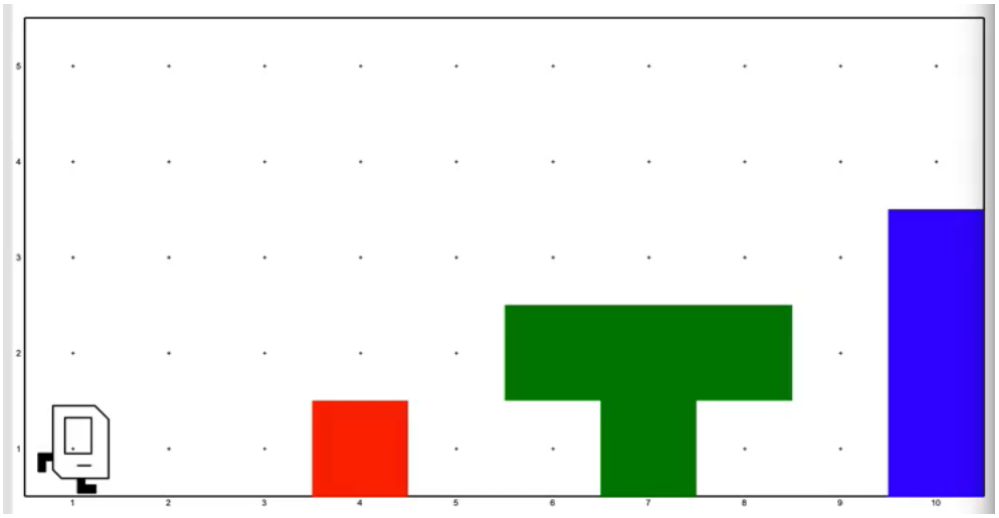


Figure 3. Karel’s World

5.2 Commanding Karel

Karel receives instructions from the human user via the command line as shown below (Figure 4). The system produces the output of the intermediary steps involved in the STP and PTR to allow us to explain the mechanics of the system so the user can understand the workings of it. The intermediary steps printed coincide with all processes described in section 4 above. The instruction in Figure 4 instructs Karel to place a diamond in its virtual "bag" on the top of the blue tower located at the extreme right of the simulation world (Figure 5).

```
What should Karel do?Put the diamond on the blue tower

Sending 'Put the diamond on the blue tower' to STP

put is an atomic verb
The predicate output is: 'put(,diamond,on(,,the blue tower))'

Sending 'put(,diamond,on(,,the blue tower))' to PTR

The destination reference is at: [10, 3]
The exact destination is at: [10, 4]
The displacement required is:  (9, 3)
```

Figure 4 .Basic sequence of instructions

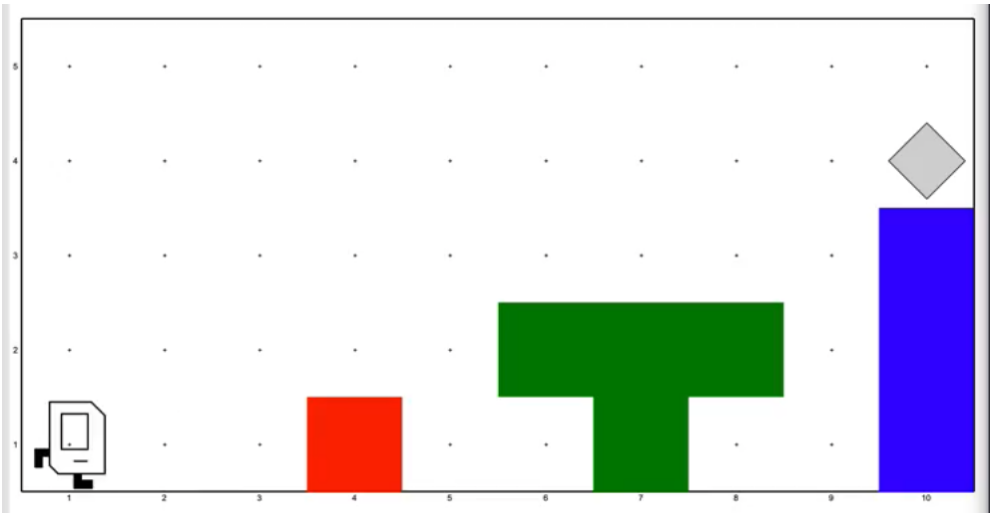


Figure 5. Effect of first instruction

In order to demonstrate Karel's ability to understand natural language instructions given and relate them to its world, we orchestrated a demonstration that instructs Karel to place multiple diamonds in a random order, and showed that Karel is able to collect all the diamonds in a random sequence independent from how they are placed (Figure 6 and 7). This proves that Karel truly understands and did not merely memorise the locations of the diamonds.

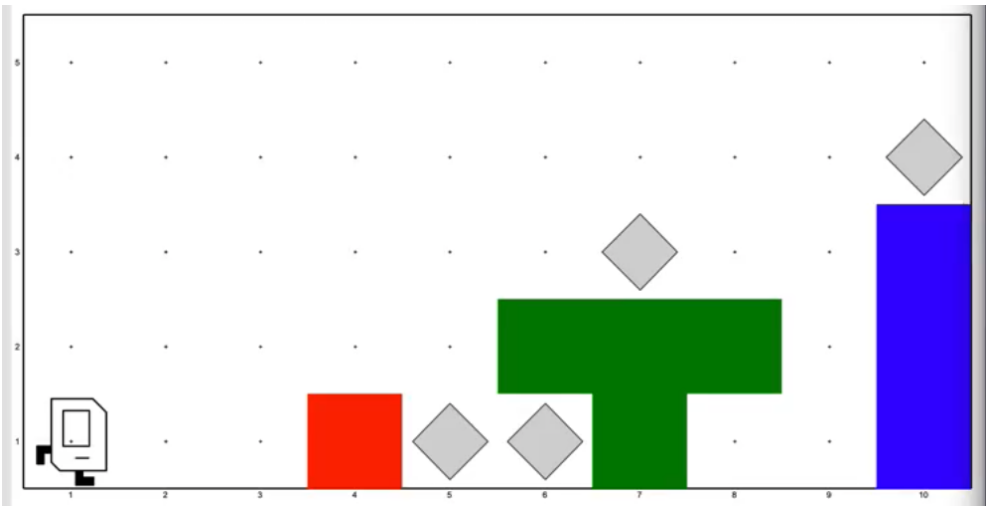


Figure 6. Karel placing diamonds in random order and positions

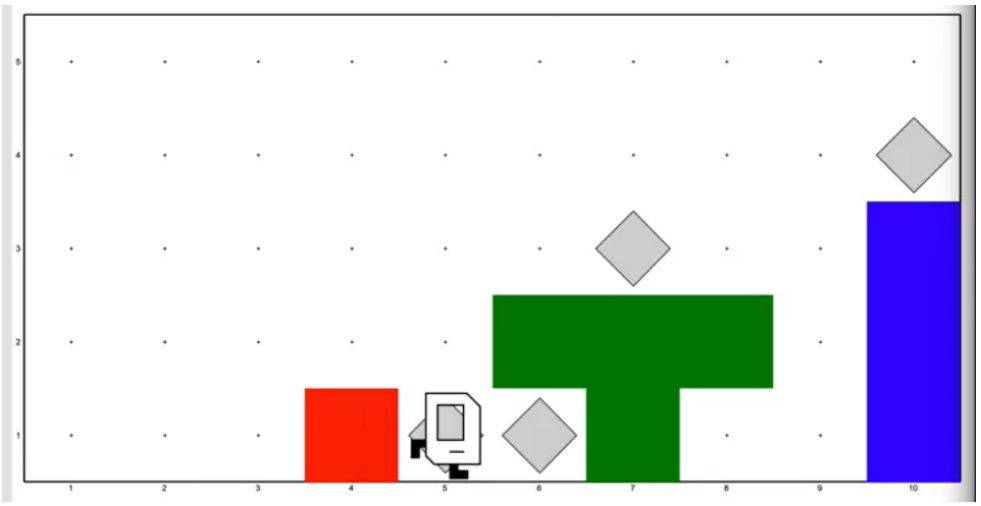


Figure 7. Karel collecting all diamonds in random order

6. Evaluation

To evaluate our system on its ability to achieve true language understanding, we first test its ability to deal with sentences of varying complexity. We then present the system to 10 different independent human subjects that have been briefed on the system, then interact with Karel via the command line interface. We then carried out a survey with two key questions to gather insights on their experience with Karel, as shown in Table 7 below. Finally, we test the system's capacity to cope with challenges that do not relate to spatio-temporal concepts and evaluate if the system's output is reasonable.

Table 7. NLU System Evaluation Survey Results

Survey Setting: Demonstrated Karel robot demo and allowed users to interact with Karel
Questions: 1. Do you think this system can truly understand you? 2. Why or why not?

Key Survey Insights:

1. Complexity and variability of sentences, grammar and vocabulary is a key contributing factor to convincing humans that the system truly understands
2. Task variability also aids in proving to humans that system truly understands

Based on our initial experiment and survey with users, we learnt that a key factor in proving that a system truly understands is through its ability to understand the same operations despite the variability of grammar and vocabulary used, as well as complexity in instructions. Another key success factor is the extensibility of the system, in terms of variation in tasks it can execute successfully. We hypothesize that we can accomplish this by extending our system to other simulation software, and also building features capable of interaction with internet applications and services, which will all be controlled via a centralised natural language interface connected to our NLU system.

7. Conclusion & Future Work

In this paper, we have introduced a systematic approach to designing a system with true language understanding. By consolidating and synthesizing concepts from natural language processing, cognitive linguistics and robotics, we have successfully demonstrated how ambiguous natural language instructions can be understood by a robotic system, demonstrated by how the robot is able to consistently carry out the right actions corresponding to it. The paper elucidates the necessary steps of analysing a natural language instruction and reconciling it with representations that a robot understands. This is done through a Sentence to Predicate conversion process which leverages a combination of cutting edge language parsers to extract the critical components of a sentence that determine its semantics; followed by a Predicate to Referent process that relates the semantics of the sentence to the physical world through knowledge of spatio-temporal relationships and actions relative to its environment.

Though our simulation's focus is on language, the outcomes are relevant to machine learning and human computer interaction in a more general sense, where findings and insights relating to NLU can be extended to other applications such as machine translation, information retrieval and sentiment analysis. These applications can benefit from improving their recognition systems to one that has true language understanding as well. All in all, these contributions will enable the development of truly intelligent robots that can achieve human-like understanding in the future.

Further work can be done to address the key determining factors that convince humans that a system truly understands, as revealed from the results of our survey evaluation above. These factors are mainly the system's ability to deal with variation in complexity and structure of the same instructions, as well as the system's ability to perform a variety of tasks from a centralised interface. Hence, in the future we plan to:

1. Extend our true language understanding system to a 3D simulation environment using the Robotic Operating System (ROS)
2. Explore the use of other simulation software packages that have native Python Application Programming Interface (API)
3. Integrate internet services and applications to our system, enabling it to carry out a variety of tasks and synthesise data from multiple sources

References

1. Alomari, M., Duckworth, P., Hawasly, M., Hogg, D. C., & Cohn, A. G. ,2017. Natural language grounding and grammar induction for Robotic manipulation commands. *Proceedings of the First Workshop on Language Grounding for Robotics*. https://doi.org/10.18653/v1/w17-2805

2. M. Alomari, D. Hogg, and A. Cohn, “Natural language learning and grounding for robotic systems,” *Cogn. Process.*, vol. 19, no. Suppl 1, pp. S14–S15, 2018, doi: 10.1007/s10339-018-0884-3.

3. Silberer, Carina, and Mirella Lapata. "Grounded models of semantic representation." In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pp. 1423-1433. 2012.

4. Siskind, Jeffrey Mark. "Grounding language in perception." *Artificial Intelligence Review* 8, no. 5-6 (1994): 207-227.

5. Siskind, Jeffrey Mark. "Grounding the lexical semantics of verbs in visual perception using force dynamics and event logic." *Journal of artificial intelligence research* 15 (2001): 31-90.

6. Chen, David L., and Raymond J. Mooney. "Learning to sportscast: a test of grounded language acquisition." In *Proceedings of the 25th international conference on Machine learning*, pp. 128-135. 2008.

7. Yu, Haonan, and Jeffrey Mark Siskind. "Grounded language learning from video described with sentences." In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 53-63. 2013.

8. Langacker, Ronald W. *Foundations of cognitive grammar: Theoretical prerequisites*. Vol. 1. Stanford university press, 1987.

9. Ho, Seng-Beng. "The atoms of cognition: A theory of ground epistemics." In *Proceedings of the Annual Meeting of the Cognitive Science Society*, vol. 34, no. 34. 2012.

10. Ho, Seng-Beng. "Operational Representation—A Unifying Representation for Activity Learning and Problem Solving." In *2013 AAAI Fall Symposium Series*. 2013.

11. H. Liu, Q. Yin, and W. Y. Wang, “Towards explainable NLP: A generative explanation framework for text classification,” *ACL 2019 - 57th Annu. Meet. Assoc. Comput. Linguist. Proc. Conf.*, pp. 5570–5581, 2020, doi: 10.18653/v1/p19-156

12. Ho, Seng-Beng, and Zhaoxia Wang. "Language and Robotics: Complex Sentence Understanding." In *International Conference on Intelligent Robotics and Applications*, pp. 641-654. Springer, Cham, 2019.

13. Ramat, P. (2011). Linguistic Typology. Linguistic Typology, 1–248. https://doi.org/10.1515/9783110859126

14. NLTK :: Natural Language Toolkit. (2021). Retrieved 27 September 2021, from https://www.nltk.org/

15. CoreNLP. (2021). Retrieved 27 September 2021, from https://stanfordnlp.github.io/CoreNLP/

16. *NetLogo*. Ccl.northwestern.edu. (2021). Retrieved 20 October 2021, from https://ccl.northwestern.edu/netlogo/

17. *ROS*. Ros.org. (2021). Retrieved 22 October 2021, from https://www.ros.org/

Appendix A: Robot Operating System (ROS) Module

Robot Operating System (ROS) is a set of open source software libraries and tools that help users quickly build and prototype robot applications ("ROS", 2021). The MoveIt! motion planning framework organises these tools and facilitates programming of custom functionalities and simulations. The framework comes with a graphical user interface (GUI) to aid in getting precise positions of robot joints and generate a sequence of movements. The simulation can then be executed in three steps: firstly, defining locations of blocks through the GUI interface; secondly, building the required sequence of instructions; lastly, execute all instructions to perform the simulation, as seen in Figure 8,9,10.

As the learning curve for ROS is relatively steep, and due to time constraints of our project, the team was unable to fully integrate our entire system with ROS. However, we were able to build a “pick and place” operation with the instruction: “Pick up the pink block and move it to the left of the orange block”, as seen in Figure 8,9,10. With more time, we can integrate our complete system with ROS capabilities.

Figure 8. Robot Operating System (ROS) 3D Simulation Module

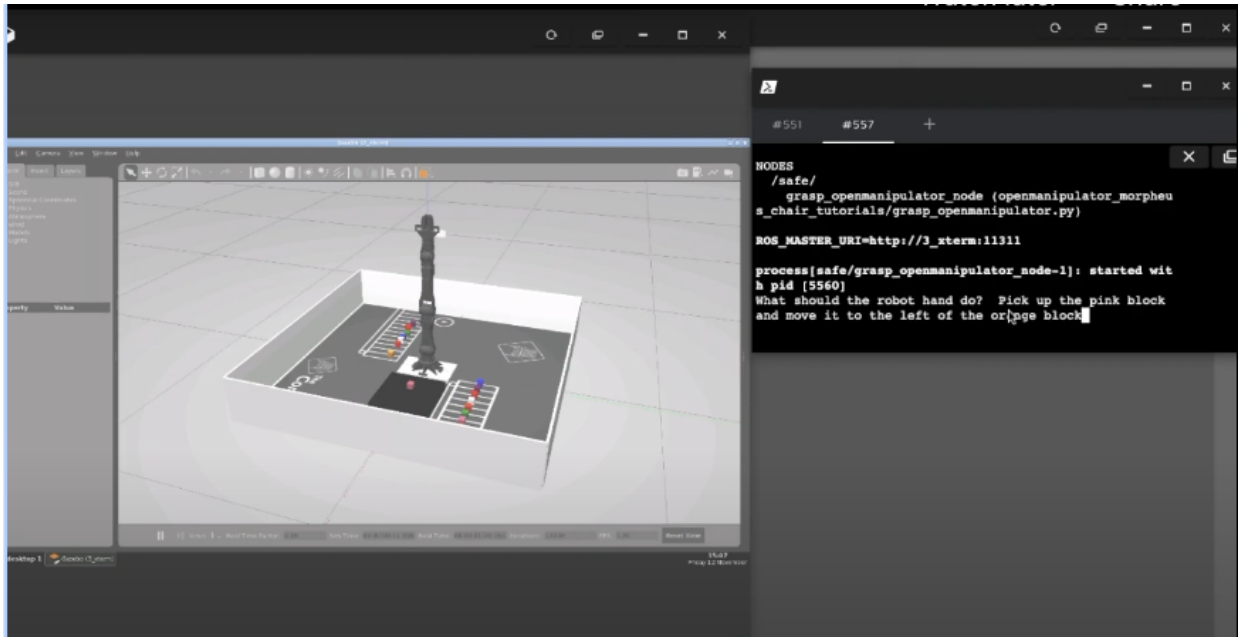


Figure 9. ROS Robotic Arm Pick Operation

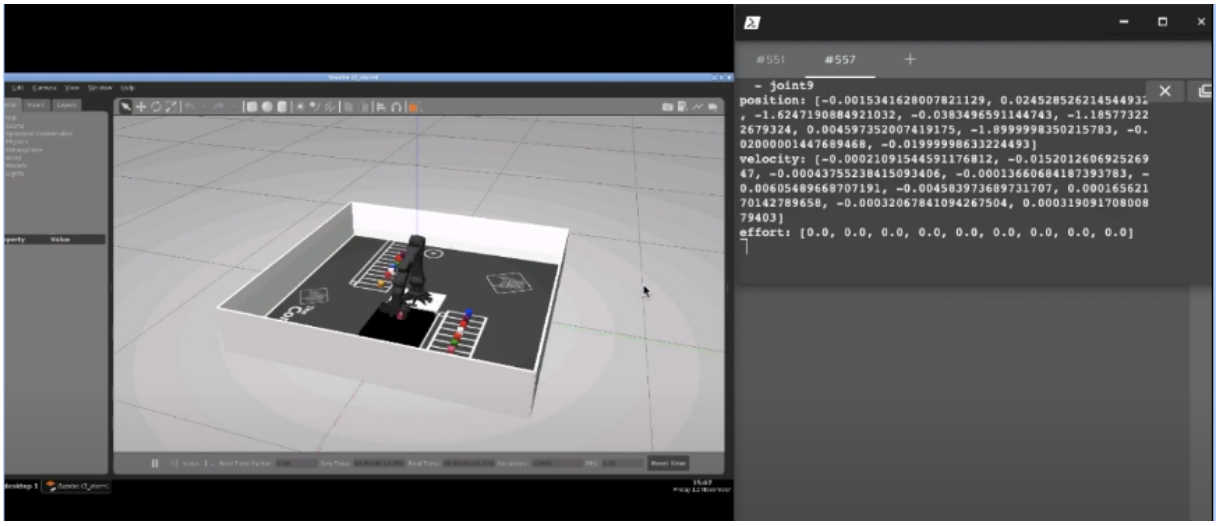
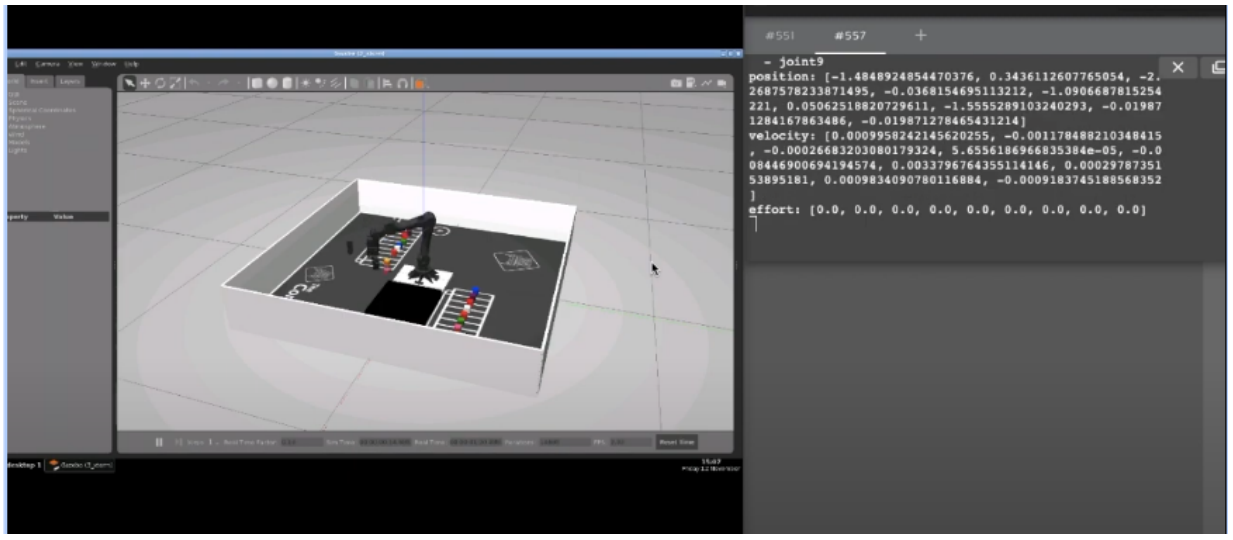


Figure 10. ROS Robotic Arm Place Operation

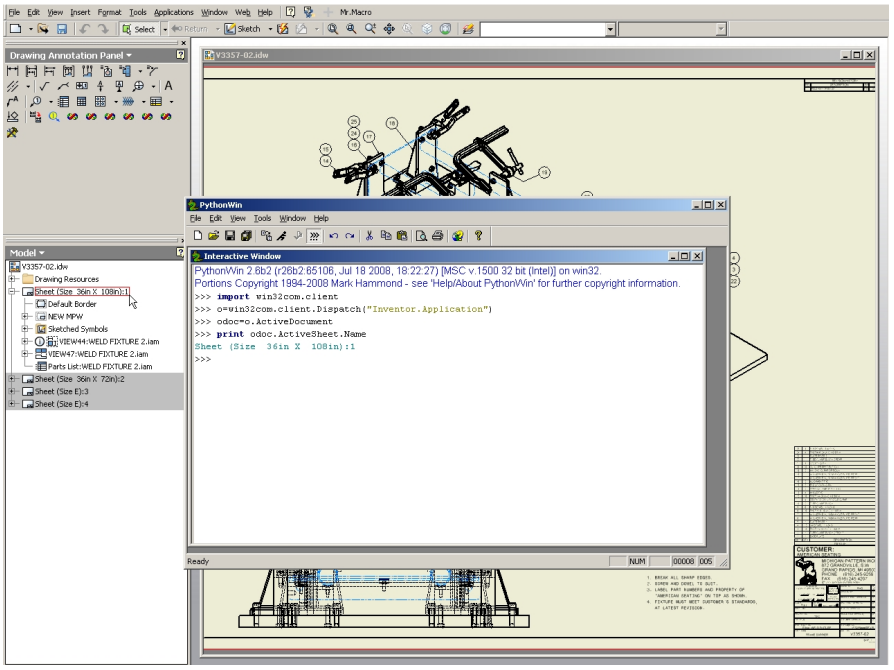


Appendix B: Incompatible Simulation Software

Autodesk Inventor

Autodesk inventor, shown in figure 11, is a computer-aided design (CAD) application used in mechanical design, simulation, visualisation. It is most commonly used for mechanical design and simulation, because users are able to construct high fidelity 3D models to aid in simulation and visualisation before building the actual physical product. Though a Python API is available, we are unable to call the custom built functions from our programs. This is because the Python API is centered around initializing and terminating simulations. Custom functions specific to the environment have been built within Autodesk Inventor software, which can only be done with VBA. Despite multiple attempts, we were unable to customise this software to meet our needs.

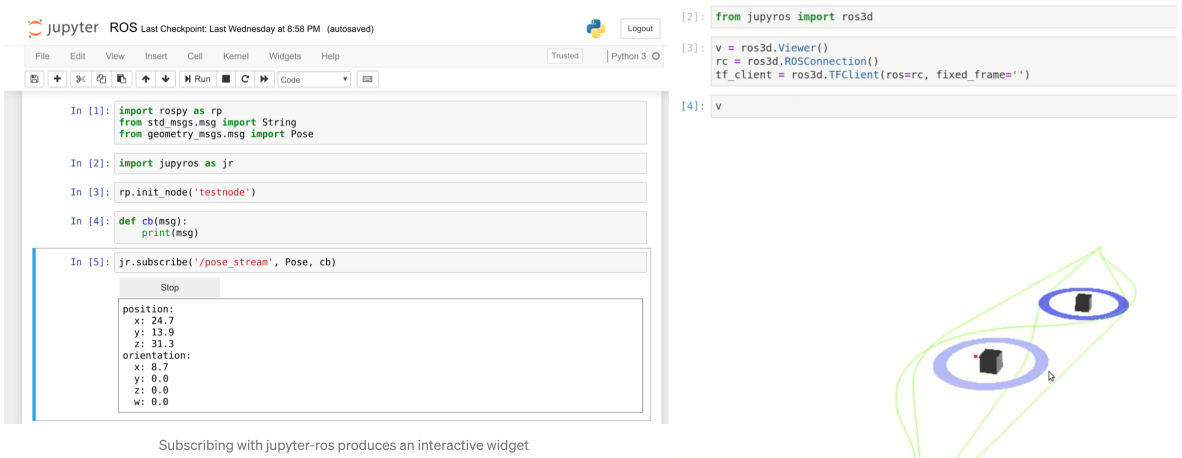
Figure 11. Autodesk Inventor Custom Functions Interface



Jupyter-ROS

Jupyer-ROS, shown in figure 12, is a suite of plugins to the Jupyter ecosystem to enable ROS to function inside Jupyter. Because Jupyter-ROS was launched only in 2019, the package is relatively new with limited documentation. In particular, there is limited extensibility, only providing APIs for visualisations relating to path planning and typical ROS pub/sub workflows, as seen in figure 12. The current visualisations available are predominantly fixed settings built for taking in predetermined parameters, and visualising how simulations differ based on different sets of parameters. At least for now, we are unable to build custom functions using Jupyter-ROS packages that resemble the native ROS functionalities as demonstrated in Appendix A.

Figure 12. Jupyter-ROS



Subscribing with jupyter-ros produces an interactive widget

NetLogo

NetLogo is a programming language and integrated development environment for agent-based modeling ("NetLogo", 2021). We have generated a 2D NetLogo environment consisting of one table, one robot claw, a yellow box, a blue box and blue ball(Figure 13). Based on the user's instructions, the claw is able to pick up the object specified by the user, move the object specified in the direction that the user commanded(left, right, up, down), and place the object in the specified location(table, floor, on top of other object, left of other objects, etc). However, in the later phase, we found out that NetLogo has limitations of 1) Unable to call custom functions from outside of NetLogo, 2) Lack of a physics engine.

Figure 13. NetLogo environment setup

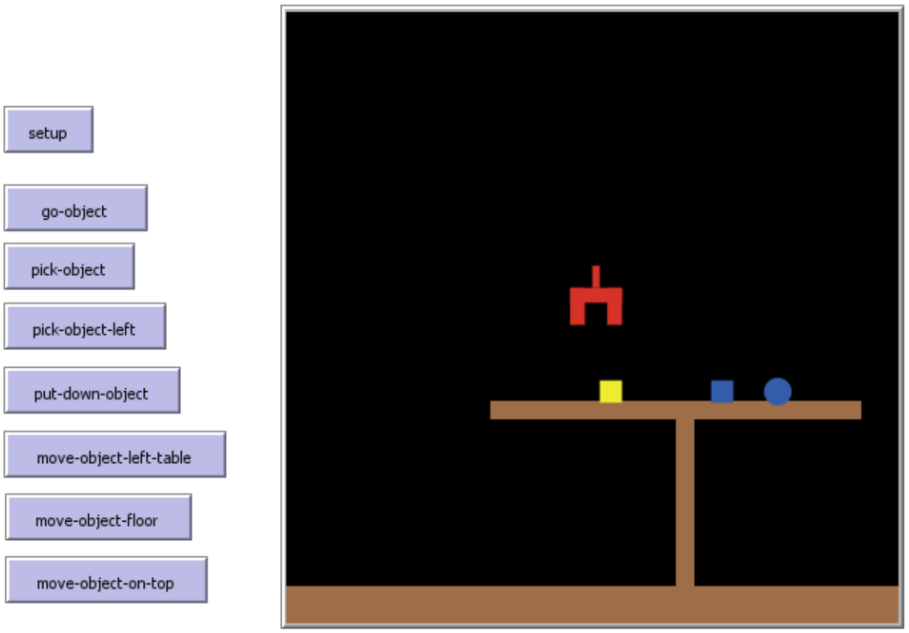
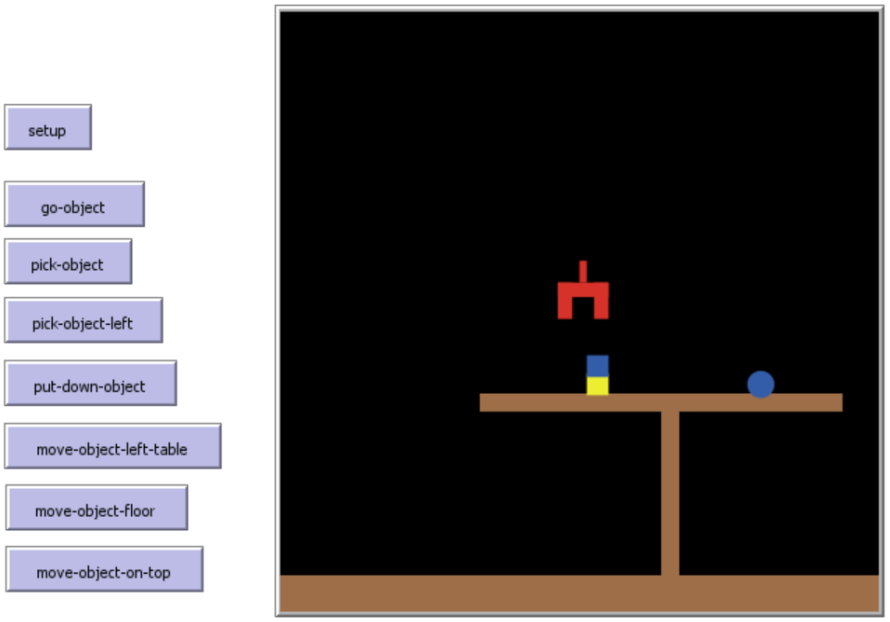


Figure 14. NetLogo simulation of put the blue box on the yellow box



Python Strings

The last method we attempted before succeeding with Stanford Karel is the use of Python strings stored in lists. A pick and place sequence is demonstrated in figure 15 below. This method failed to work because the laborious programming of printing the update of states at each step is not scalable. This is when we learnt that having a built-in physics engine able to visualise the effects of gravity, as well as display the change in state of the environment at each intermediary step is a key feature required when choosing a simulation software to integrate with our NLU system.

Figure 15. Python String method