

# IS470 Guided Research in Information Systems

## **True Language Understanding for more human-like machines**

By: Brandon Ong

01

## Research Problem

Motivation and existing work

02

## Research Approach

Proposed methodology

03

## Adopted Methodology

Detailed steps to achieve  
research goal

04

## Conclusion

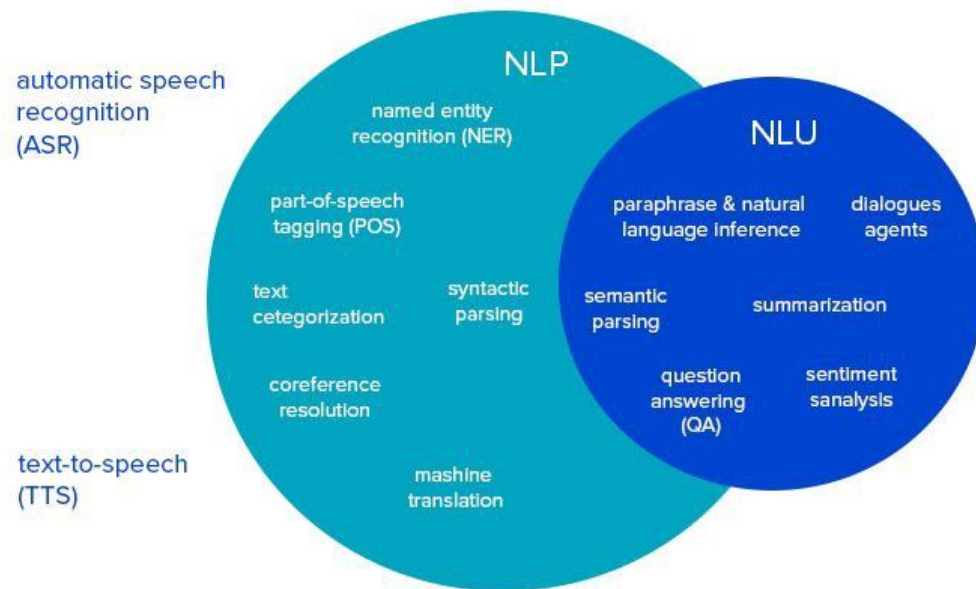
Evaluate and future work

# Research Problem

## What is true natural language understanding (NLU)?

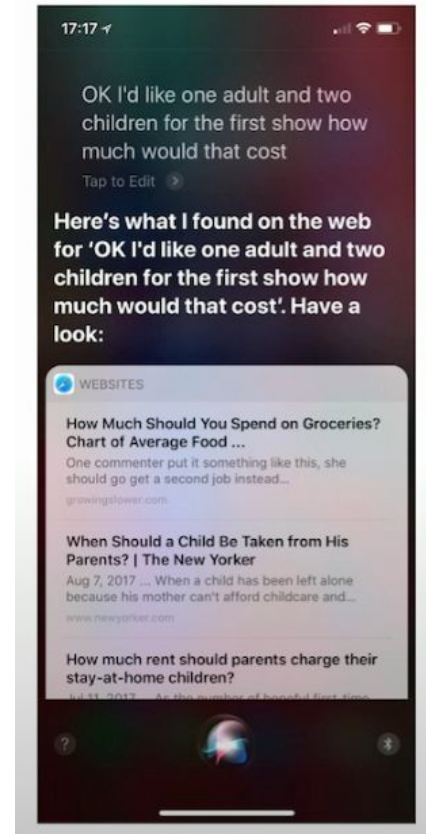
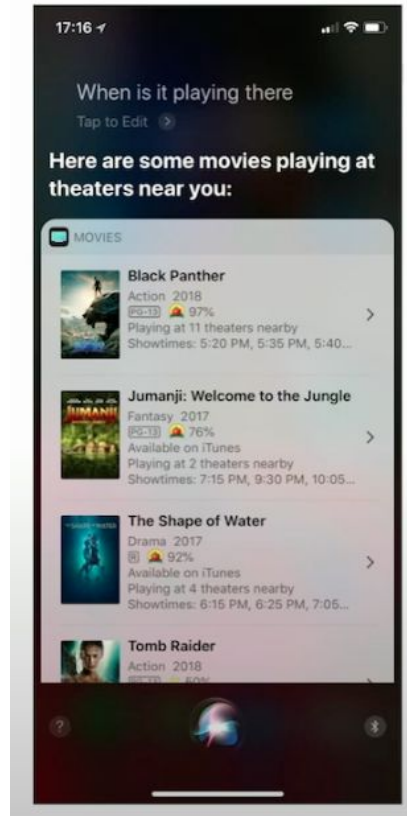
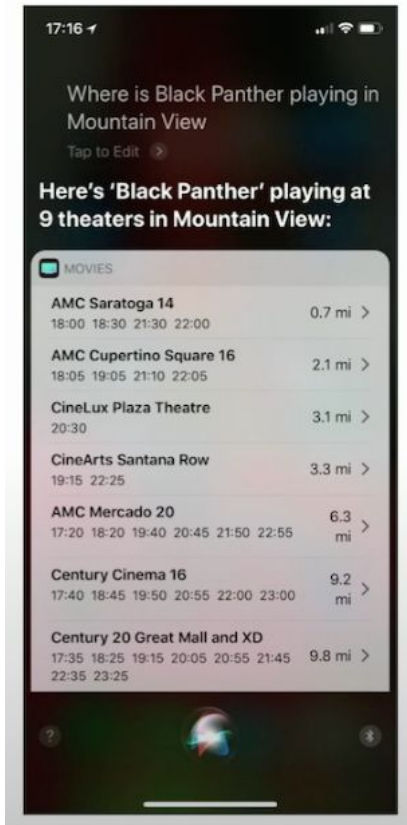
NLU goes beyond recognition to determine a user's intent. Enable computers to understand and interpret human language for better human computer interaction (HCI)

**Model human understanding by codifying common sense:** When I think of moving something, it requires picking it up, moving to the destination and placing it down



# Relevance - Enhancing HCI

## Chatbots - Conversational Interface



Can be further abstracted to machine translation, information retrieval, sentiment analysis

## Why NLU?

Addresses **limitations** of recognition-based systems that power NLP today (deep learning-based systems like attention and transformers)

### Limitations:

1. Context dependence: Where are you from?

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.

“*He*” refers to Brandon. “*It*” refers to the bank.

1. Winograd's SHRDLU (1973) - Research

- A robotic arm in a simulated world able to carry out human instructions



2. Cyc, pronounced "Sike" (2017) - Industry

- Applied NLU concepts to use cases in industries like healthcare and logistics



## Limitations and criticisms

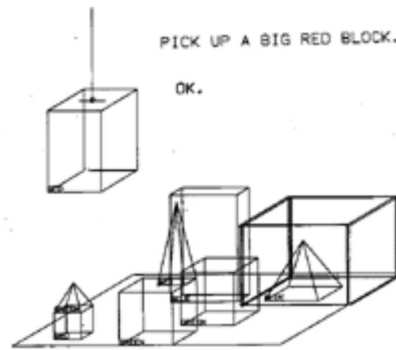
- Scalability issues because required all syntax and semantics (built with Lisp)
- Selection of prototypical cases to build from
- Focus on answering over acting

## Consequences

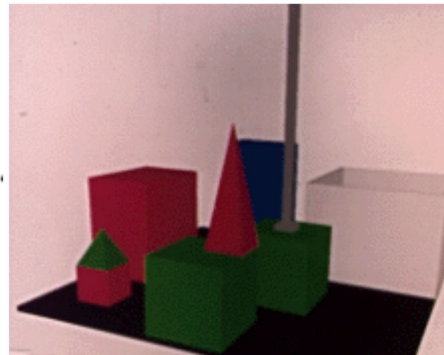
- Less attention to further develop concept in research space
- **Possible explanations:** Deep learning trends; require interdisciplinary expertise; different focus of related industries - Humanities (cognitive linguistics), Robotics (mechanical optimisation)

## Dialog with SHRDLU:

- **Person:** PICK UP A BIG RED BLOCK.
- **Computer:** OK. (does it)
- **Person:** GRASP THE PYRAMID.
- **Computer:** I DON'T UNDERSTAND WHICH PYRAMID YOU MEAN



Original screen display



Later color rendering (Univ. of Utah)



# Research Approach

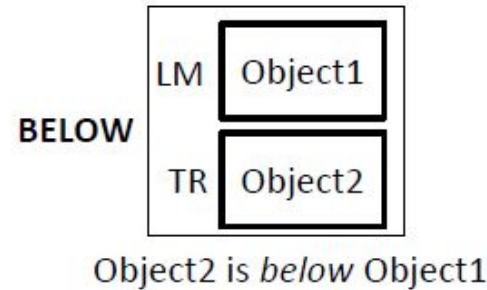
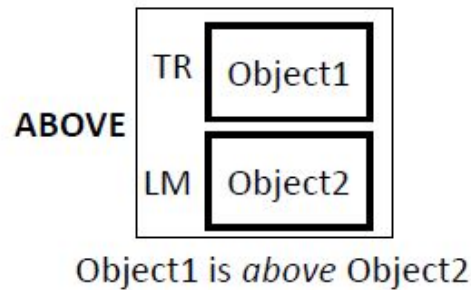
# Our approach - how different?

---

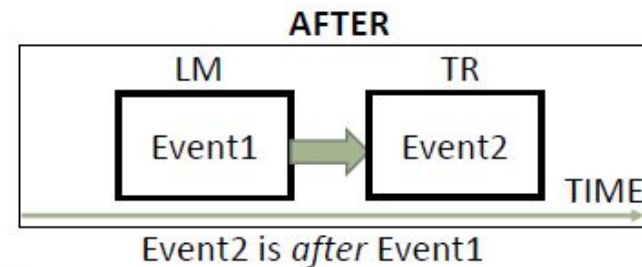
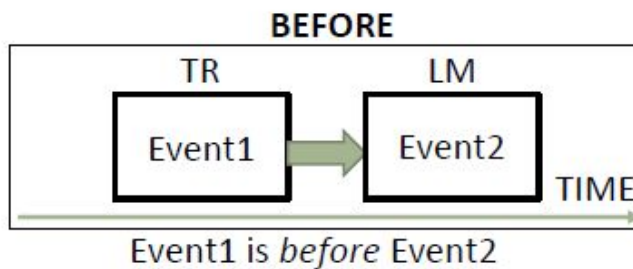
- 1. Knowledge base and representations used**
  - a. Spatio-temporal concepts
  - b. Landmark-trajectory relationship
  - c. Action representations
- 2. Parsing and processing process**
  - b. Combination of StanfordCoreNLP and NLTK
  - c. Two-step processing process:
    - i. Sentence to Predicate (STP)
    - ii. Predicate to Referent (PTR)

# Spatial relations

- Definition of trajector (subject) and landmark (object) need to be standardised



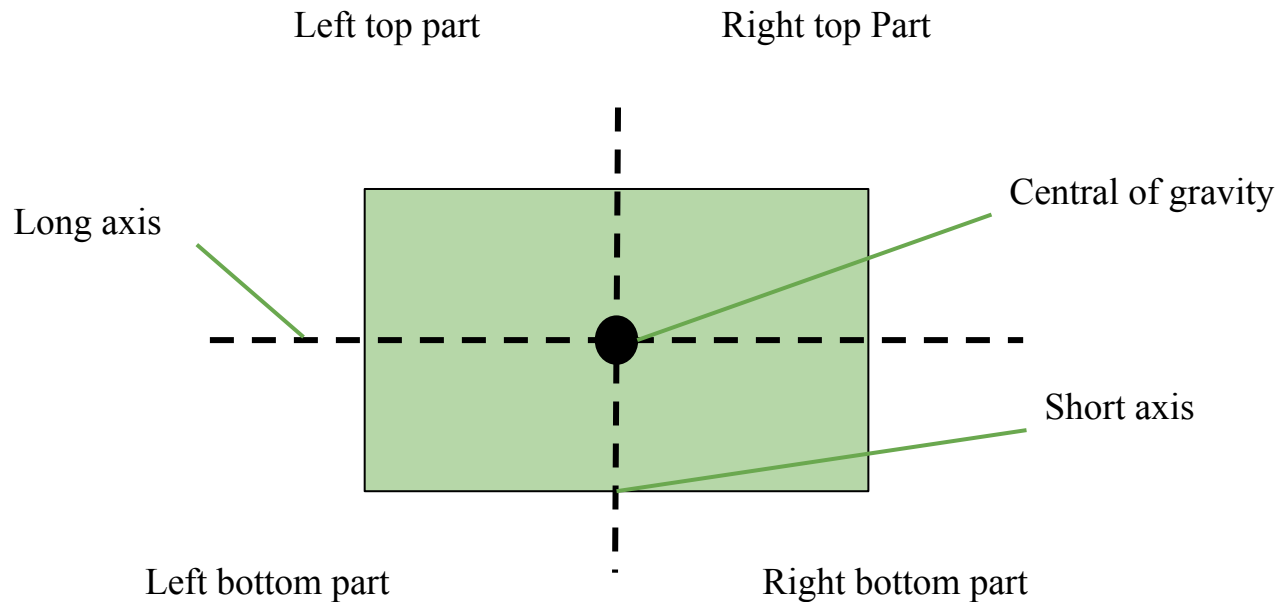
(a)



(b)

# Spatial relations

- Spatial relations relative to a standardised central of gravity (CG)



Complex action relations will be broken down into atomic action concept. We then combine action relations and spatial relations to form predicates.

## Complex, non-atomic action

Robot **carry** the ball

>> **lift**(robot,ball,**hold**(robot,,))

## Atomic action

move the ball to the right of the box

>> move(, ball, right(,,the box))

\*If no subject is indicated, it will automatically assumed to be the robot

\*\*The predicate is in a format of Predicate(Subject, Object, Goal). if the subject is indicated, the nested predicate will inherit from the parent. Hence it is represented as an empty parameter in right(,,the box)

# Proposed System

Aim to build a system that **understands complex instructions** by leveraging knowledge of **atomic concepts** that a machine already understands.

This is achieved through TWO key mechanisms:

Sentence to Predicate (STP) converter	Predicate to Referent (PTR) converter
STP's key role is to parse the instruction into a standardised format that the PTR understands	PTR will parse the STP's output into the key components a robot requires to carry out the instruction

# Our contributions

---

1. Demonstrate how to successfully **parse** a natural language instruction and successfully **command** a robot to execute it
2. Demonstrate the **accuracy and limitations** of cutting edge language parsers
3. Highlight components that will be **critical to the success** of a system that achieves true language understanding

# Adopted methodology



# Planned Timeline

Week Number	Weekly Project Goals
1	Project introduction and deep literature review
2	Deep literature review
3	Research grammar parsers
4	Project proposal
5	Prototyping STP using grammar parsers
6	Integrating grammar parser with grounding concepts
7	Mid Term evaluation
8	Learning AutoCAD
9	Prototyping with AutoCAD
10	Build PTR with AutoCAD
11	Integrate PTR with STP
12	Increase complexity of integrated system
13	Evaluate system externally and reiterate on feedback
14	Presentation
15	Term Paper

# Sentence to predicate converter

We have used Python predominantly for writing programs for the sentence to predicate converter and predicate to referent converter. We have also used Natural Language Toolkit (NLTK) and Stanford Core NLP for performing sentence grammar parsing.

A natural language parser is a program that works out the grammatical **structure of sentences**, for instance, which groups of words go together (as "phrases") and which words are the **subject** or **object** of a verb.

Sentence = 'Pick up the ball'

NLTK `[('Pick', 'NNP'), ('up', 'RP'), ('the', 'DT'), ('ball', 'NN')]`

Stanford Core NLP 

```
(ROOT
  (S
    (VP (VB pick)
      (PRT (RP up))
      (NP (DT the) (NN ball)))))
```

# Sentence to predicate converter

## Linguistic typology

subject–verb–object (SVO) is a sentence structure where the subject comes first, the verb second, and the object third. Languages may be classified according to the dominant sequence of these elements in unmarked sentences

SVO is the second-most common order by number of known languages, after SOV.

Subject–verb–object languages almost always place relative clauses after the nouns which they modify and adverbial subordinators before the clause modified

‘Can you lift the box’

‘Can you lift the box by 5 cm’

‘Tom throw the ball across the hall’

Word Order	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, German

We added a good number of verb and spatial groundings that we plan to include in our processor based on what we think are commonly used in instructions.

<u>Verb Grounding</u>		
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

<u>Spatial Grounding</u>		
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

# Sentence to predicate 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)))  <u>Referent</u> If not mentioned, right of the box will by default be a 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)))
"move the ball above the box"	above	move	move(, ball, above (, , the box))
"align the claw to the ball"	to	align	align(, claw, to(, , the ball))
"lift the ball"	-	lift	lift(ball,)

# Sentence to predicate converter

We will first convert the instructions to corresponding temporal structures. This is just a demonstration, the system will be able to understand the instruction and carry out action in any orders of the instructions. This is only an example that showcased a simple environment, there could be more objects in the environment

**‘Pick up the box’**

`pick(box,up(, the box))`

**‘Put the box to the right of the table’**

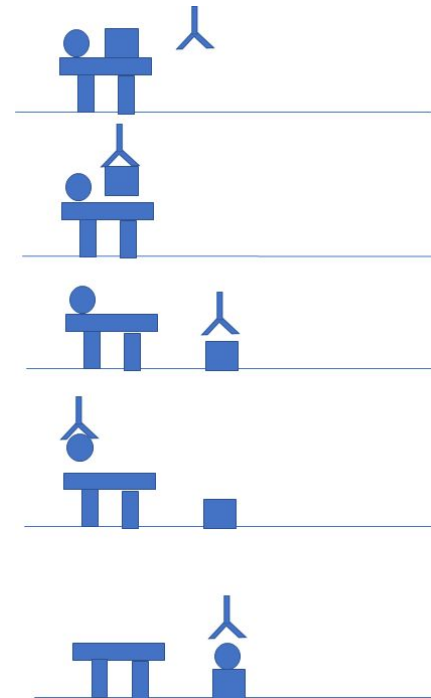
`put(box,to(,right(, of the table)))`

**‘Pick up the ball’**

`pick(ball,up(, the ball))`

**‘Put the ball above the box’**

`put(ball,over the box)`



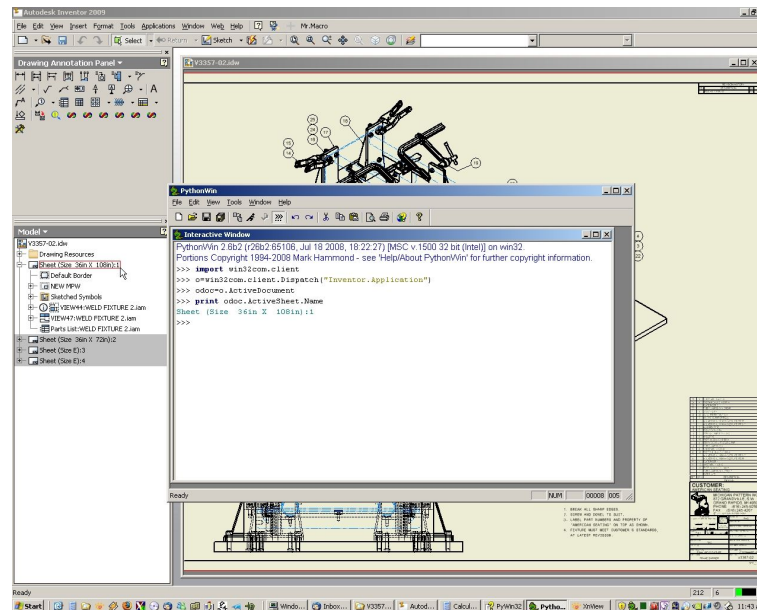
## Overview

- We planned to use AutoCAD initially, but was unable to do so eventually
- We experimented with Jupyter-ROS, Netlogo, python strings, and finally Stanford-Karel before achieving a workable prototype
- Stanford-Karel is a 2D simulation software, but is able to successfully demonstrate the key mechanisms necessary for a system that achieves true NLU
- In parallel learnt and researched the use of Robot Operating System (ROS)

**Limitation:** Python API is available, but not designed to build custom functions. Functionality mainly to start/stop simulation and create parts. Custom functions possible but API written in VBA within AutoCAD

**Critical issue:** No “interface” for calling custom functions built in VBA. Unable to call custom functions from Python components

**Possible explanations:** Software used primarily for 3D modelling and simulation

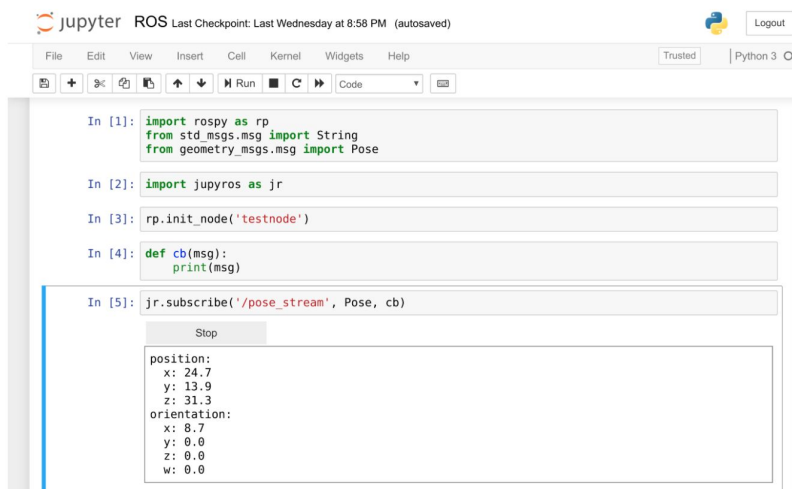




**Limitations:** Jupyter-ROS (Python package) has limited extensibility. Limited built-in visualisations. Mainly built for taking in fixed parameters and visualising changes based on inputs

**Critical Issue:** Unable to build custom functions that can be called from our python programs

**Possible explanations:** Package focus on path planning, follow conventional ROS setting using pub/sub patterns



The screenshot shows a Jupyter Notebook titled "jupyter ROS Last Checkpoint: Last Wednesday at 8:58 PM (autosaved)". The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for file operations, cell navigation, and execution. The code cells contain the following Python code:

```
In [1]: import rospy as rp
        from std_msgs.msg import String
        from geometry_msgs.msg import Pose

In [2]: import jupyros as jr

In [3]: rp.init_node('testnode')

In [4]: def cb(msg):
        print(msg)

In [5]: jr.subscribe('/pose_stream', Pose, cb)
```

Below the code cells, an interactive widget displays the output of the subscription:

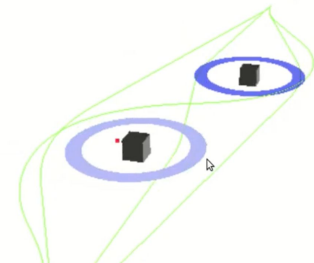
```
position:
  x: 24.7
  y: 13.9
  z: 31.3
orientation:
  x: 8.7
  y: 0.0
  z: 0.0
  w: 0.0
```

Subscribing with jupyter-ros produces an interactive widget

```
[2]: from jupyros import ros3d

[3]: v = ros3d.Viewer()
      rc = ros3d.ROSConnection()
      tf_client = ros3d.TFClient(ros=rc, fixed_frame='')

[4]: v
```



## PTR requires TWO key components:

### 1. Predicate parsing

Extracting the key components of the predicate needs to be standardised and generalisable

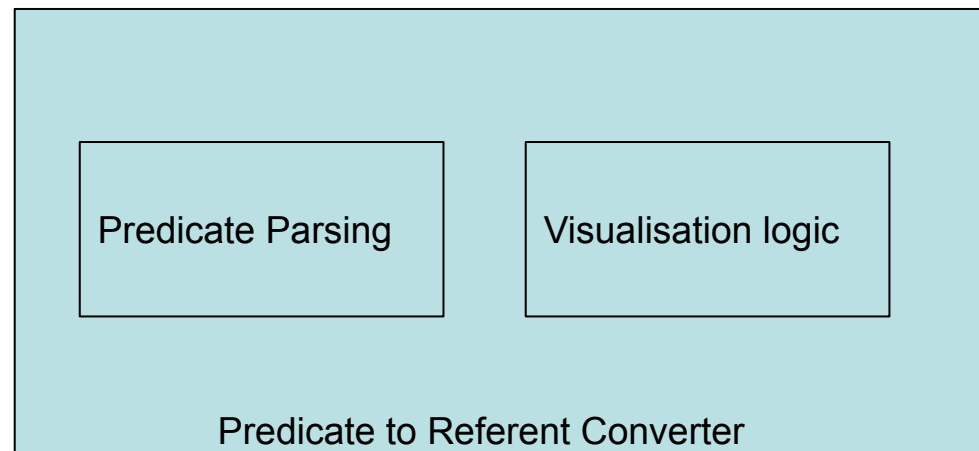
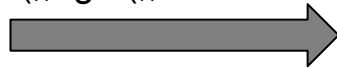
This will affect the format of the STP that we output

### 2. Visualise

b. Custom functions built must be able to be called by external programs

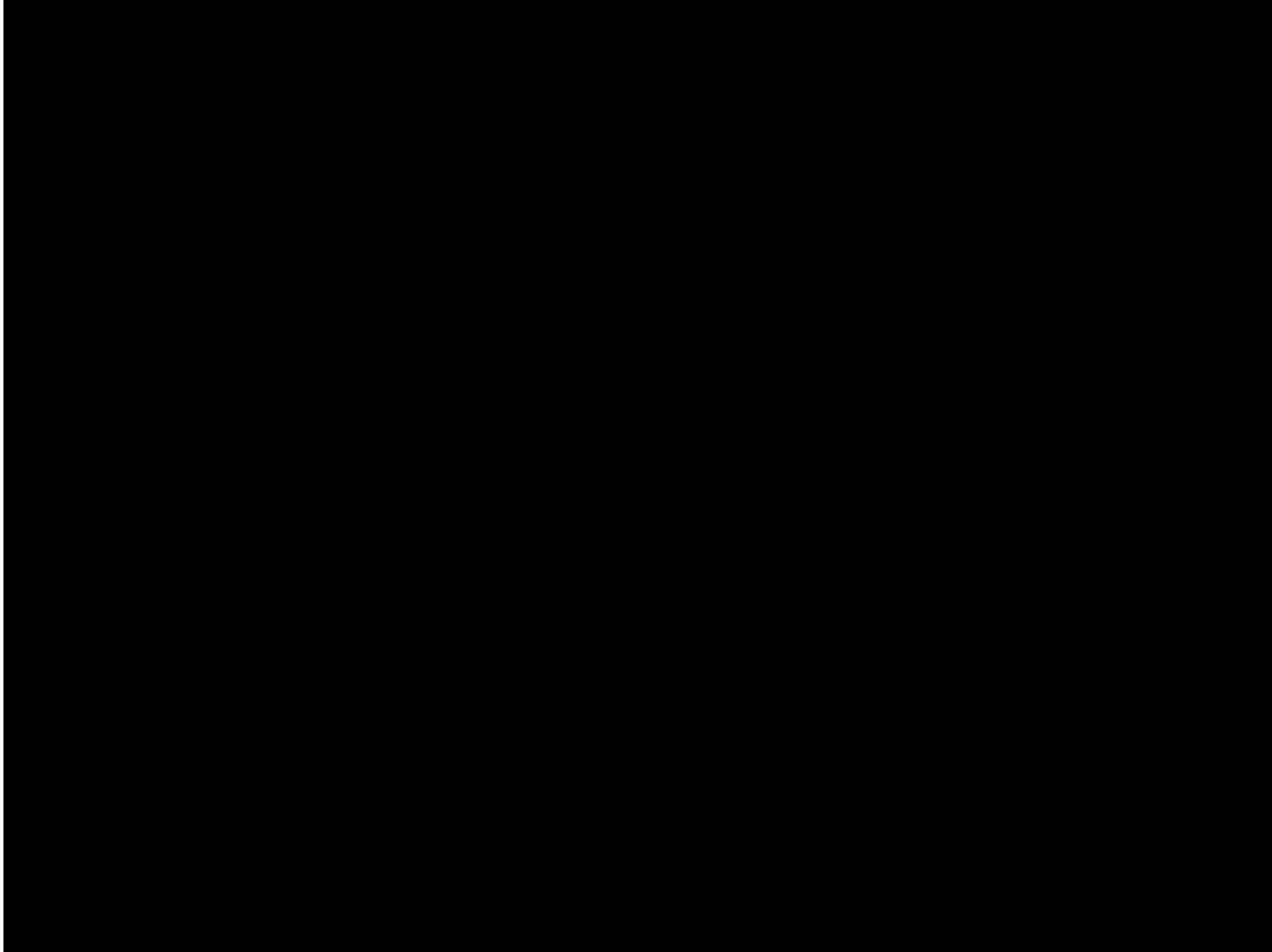
c. Having a built-in physics engine is critical for 3D visualisation, hardcoding physics effect should not be done

Predicate  
put(box,to(,right(, of the table)))



# Stanford-Karel Demo

---



# Eventual Timeline

Week Number	Weekly Project Goals
1	Project introduction and deep literature review
2	Deep literature review
3	Research grammar parsers
4	Project proposal
5	Prototyping STP using grammar parsers
6	Integrating grammar parser with grounding concepts
7	Mid Term evaluation
8	AutoCAD R&D
9	ROS R&D
10	NetLogo R&D
11	Python string method R&D
12	Stanford Karel R&D
13	Integration with STP, testing and reiterations
14	Presentation
15	Term paper

# Conclusion

Survey of 10 independent users: Would you believe this system understands you? Why or why not?

## **Common TWO insights**

1. A variation in **sentence structure** will convince me that the system truly understands
2. The variation of **tasks** the machine can respond appropriately to will convince me that it understands human language

## **1. Translate visualisation from 2D to 3D**

Our two-dimensional (2D) demonstration with Karel is a successful proof of concept (PoC) of the end-to-end process of what an NLU system requires. We can translate the concepts into a 3D simulation once we figured out how to operate a 3D environment. Our modular system allows us to do this easily

## **2. Expand system capabilities for more use cases**

This is a key factor in demonstrating to humans that a system truly understands

# Appendix



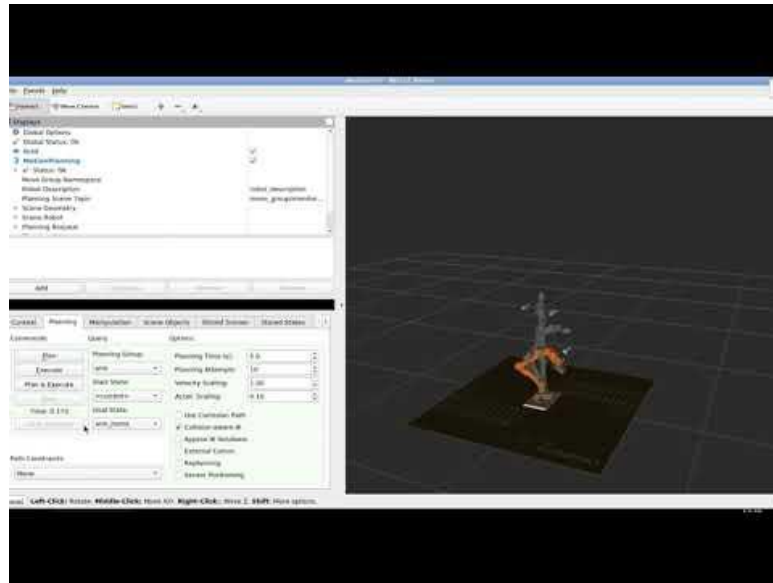
ROS requires setup of packages and dependencies.

**Key components used:**

- Moveit! package that will handle kinematics of simulation
- Robot arm can be programmed through python script and called externally
- GUI to calculate precise locations in (x,y,z) coordinates

# ROS GUI - Coordinate setting

Moveit! GUI - to get position of robot joints




Group joints of goal state	
Joint Name	Value
id_1	0°
id_2	58°
id_3	-115°
id_4	0°
id_5	-83°
id_6	0°
id_7	0°


Python Script - to generate sequence of movements

```
joint_position_init = [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, -1.9, 0, 0]
joint_position_home = [0.006135923322290182, 0.13499031960964203, -1.9067381620407104, -0.03681553900241852, -1.323825478553772, 0.003067961661145091, -1.9, 0, 0]
pink_open = [-0.0015339808305725455, 0.02454369328916073, -1.9726994037628174, -0.03834952041506767, -1.1857671737670898, 0.004601942375302315, -1.9, 0, 0]
pink_close = [-0.0015339808305725455, 0.02454369328916073, -1.9726994037628174, -0.03834952041506767, -1.1857671737670898, 0.004601942375302315, -1.0, 0, 0]
```

1. Predefine locations of blocks
2. Append required sequence to an array
3. Loop through the array to get simulation



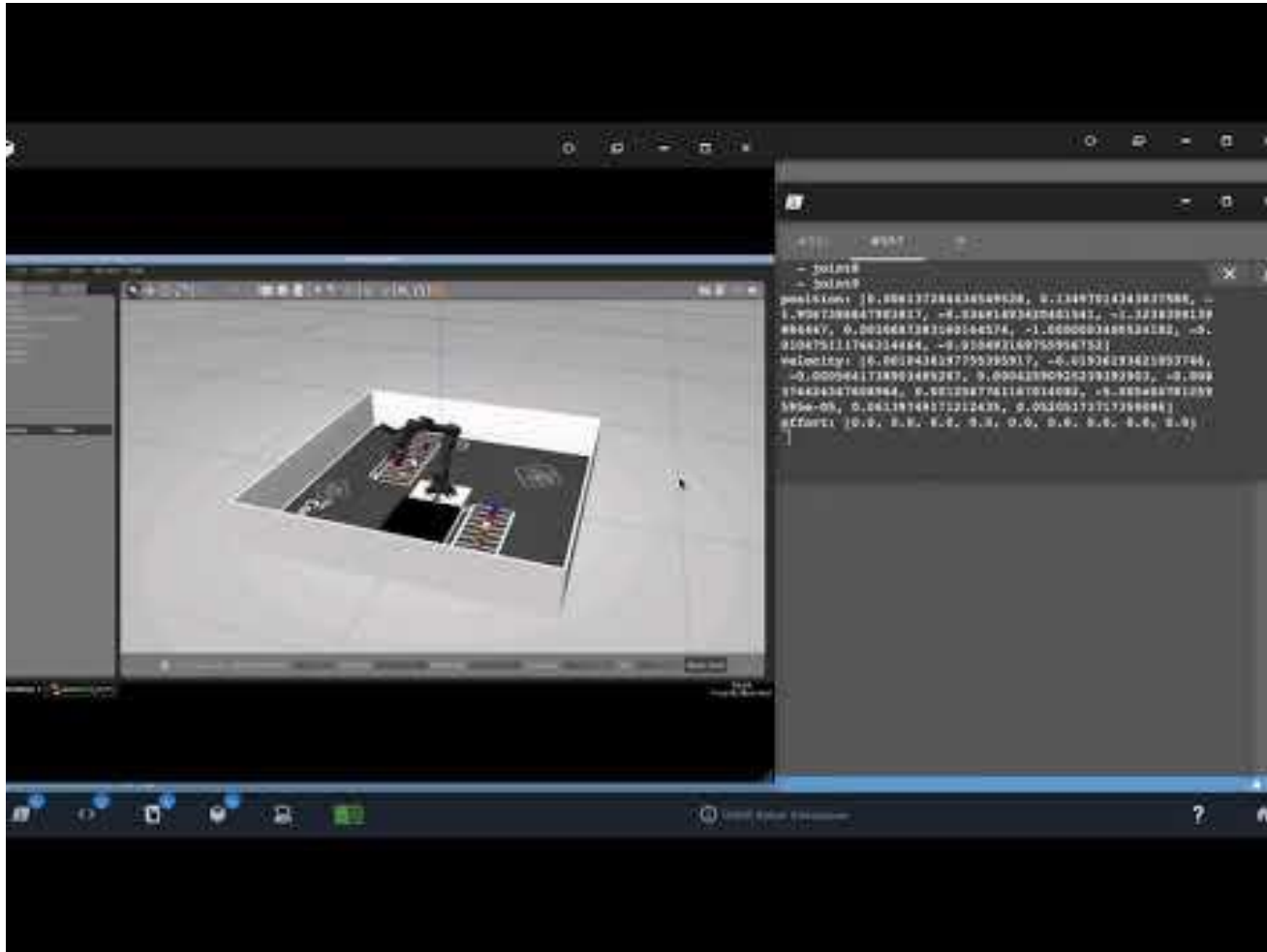
```
joint_position_sequence = []
joint_position_sequence.append(joint_position_init)
joint_position_sequence.append(joint_position_home)
joint_position_sequence.append(pink_open)
joint_position_sequence.append(pink_close)
joint_position_sequence.append(move_up_test)
```



```
for joint_position_array in joint_position_sequence:
    openman_obj.move_all_joints(joint_position_array)
    time.sleep(movement_speed)
```

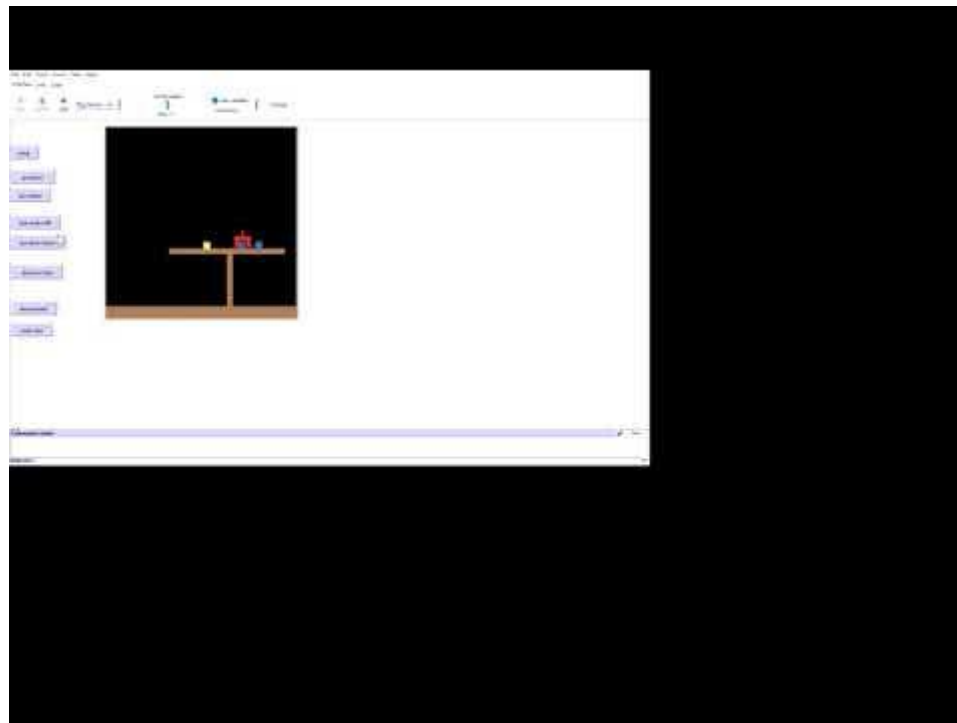
**Instruction:** “Pick up the pink block and move it to the left of the orange block”

- **pick**(,pink block,**to**(, , **left**(, , orange block))



NetLogo is a programming language and integrated development environment for agent-based modeling and we are using it for 2D visualisation of our environment.

**Limitation:** Unable to call custom functions from outside of NetLogo. Lack of physics engine



**Questions?**

**Thank You**