Multimodal Semantics for Affordances and Actions

Lecture 4: Communicating in Multimodal Common Ground

James Pustejovsky and Nikhil Krishnaswamy

ESSLLI 2022 Summer School

Galway, Ireland

August 8-19, 2022

Course Outline

- Monday: Components of Multimodal Communication
- Tuesday: Modeling Human-Object Interactions
- Wednesday: Modeling Multimodal Common Ground
- Thursday: Communicating with Multimodal Common Ground
- Friday: Reasoning with and about Affordances

Thursday's Outline

- Overview of VoxWorld platform implementation
- Communicating with VoxWorld agents
- Unimodal communication
- Multimodal communication
- Correction and clarification

What Makes an Agent?

- Perceives through sensors and acts through actuators
- Epistemic point of view from which it observes the world
- Virtual world is **mode of presentation**, allows observer to see what agent does
- Embodied agents add new dimensions to human/agent interactions
- Must recognize and interpret inputs in multiple modalities (e.g., gesture, speech, gaze, action)
- Solving these problems has driven development of **VoxWorld**: a platform for multimodal agent behaviors

VoxWorld Platform

- VoxML modeling language and VoxSim event simulator
- Events composed of subevent semantics that decompose into minimal primitive set
- Objects encoded with **habitat** and **affordance** properties
- Relations sample from distributions under constraints
- Event, relations, and objects composed at runtime
- Multiple semantic theories may be mutually compatible
    - **Problem**: Computationally difficult to implement

VoxWorld Platform

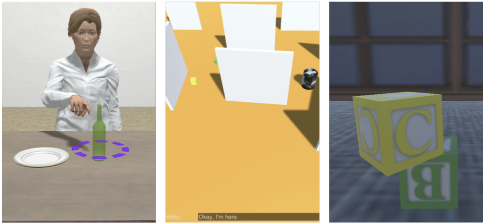


Figure: 3 VoxWorld agents

- VoxWorld: Built on Unity game engine
- Accommodates qualitative calculi, machine learning inputs
- Simulated environment operationalizes and unifies multiple frameworks
- Primary language: C#
    - General-purpose, multi-paradigm

VoxWorld Platform

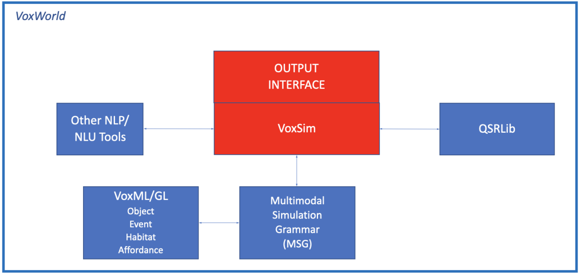


Figure: VoxWorld generic architecture

Architecture as depicted:

- Handles consumption/composition of language, multimodal inputs
- Does not explicitly handle common ground or dialogue state

1. **Blackboard architecture**
    - Originally developed for HEARSAY NLU system (Erman et al, 1980)
    - Strongly-typed key value store in singleton design pattern
    - Subscribe functions to keys, which trigger upon key changes
    - Stores common ground-relevant information
2. **Pushdown automaton**
    - Tuple of states Q, inputs Σ, stack symbols Γ, transition relation δ
        - Initial state $q_0 \in Q$, Initial stack symbol $Z \in \Gamma$, accepting states $F \subset Q$
    - PUSH, POP, REWRITE stack operations
        - Add: FLUSH, POPUNTIL
    - Use blackboard state as stack symbol
        - Evaluate stack symbols as functional satisfiable predicates

VoxWorld Platform

Two ideas from classical AI

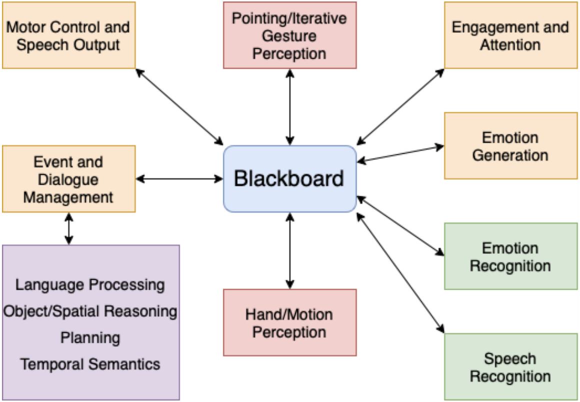


Figure: Sample blackboard knowledge inputs

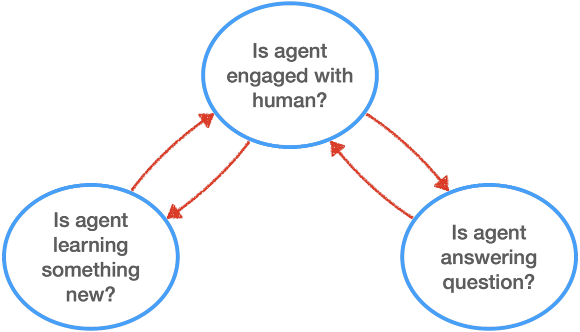


Figure: High-level pushdown automaton

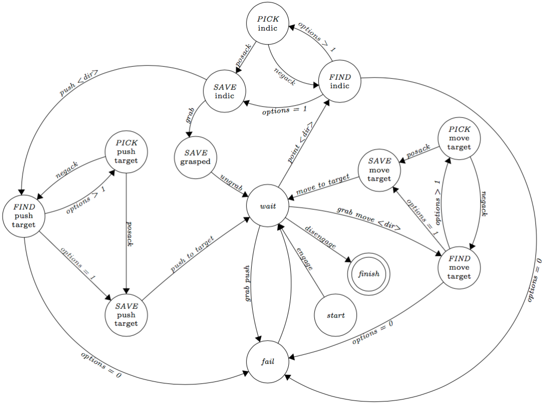
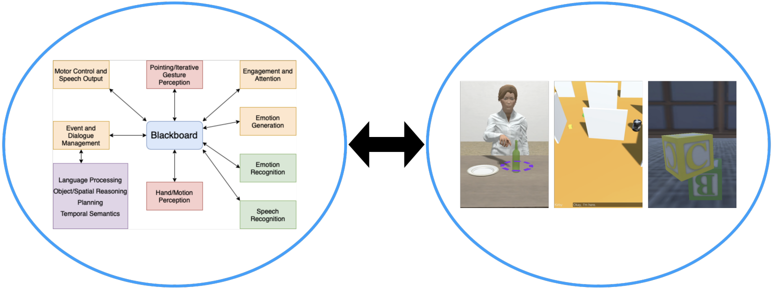


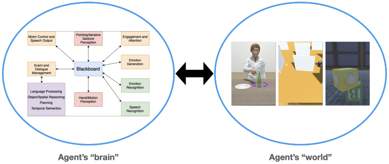
Figure: Low-level pushdown (c. 2017-8)

Two ideas from classical AI



Agent's "brain" **Agent's "world"**

VoxWorld Platform

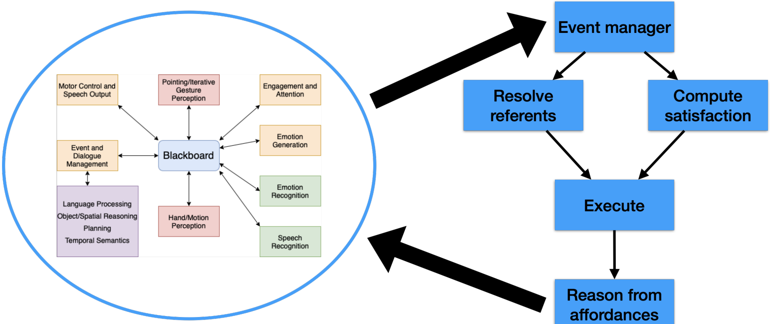


- "Brain" updates change internal information state
- "World" updates enact change in the environment
- Blackboard/PDA ("brain") may trigger executable functions, incl. agent moving items in the environemnt ("world")
- Actions in the world may prompt updates in common-ground
  - Human can infer what the agent does/doesn't know
  - Agent may infer things about the human!

VoxWorld Platform

VoxML operationalized

High-level flow of control:

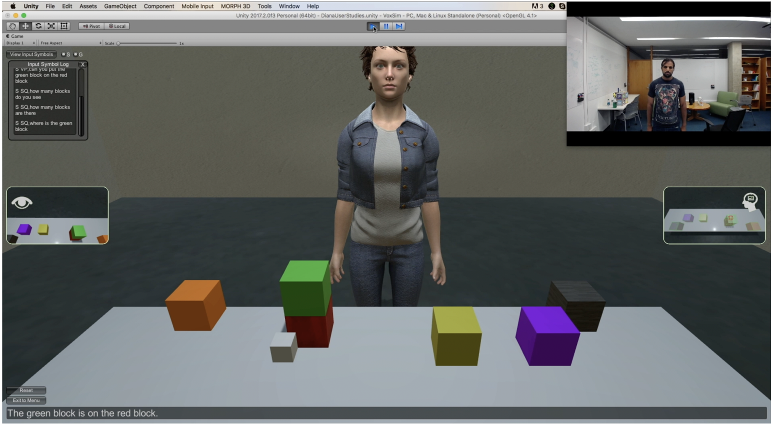


Affordances

$$
\left[\begin{array}{l}
\textbf{block} \\
\text{AFFORD\_STR} = \left[\begin{array}{lcl}
\mathrm{A}_1 & = & H_{[2]} \rightarrow [put(x, y, on([1]))] \\
 & & support([1], y) \\
\mathrm{A}_2 & = & H_{[2]} \rightarrow [grasp(x, [1])] \\
 & & hold(x, [1]) \\
\mathrm{A}_3 & = & H_{[2]} \rightarrow [lift(x, [1])] \\
 & & hold(x, [1]) \\
\mathrm{A}_4 & = & H_{[2]} \rightarrow [ungrasp(x, [1])] \\
 & & release(x, [1])
\end{array}\right]
\end{array}\right]
$$

VoxWorld Platform

VoxML operationalized



▸ Link

VoxWorld Platform

VoxML operationalized

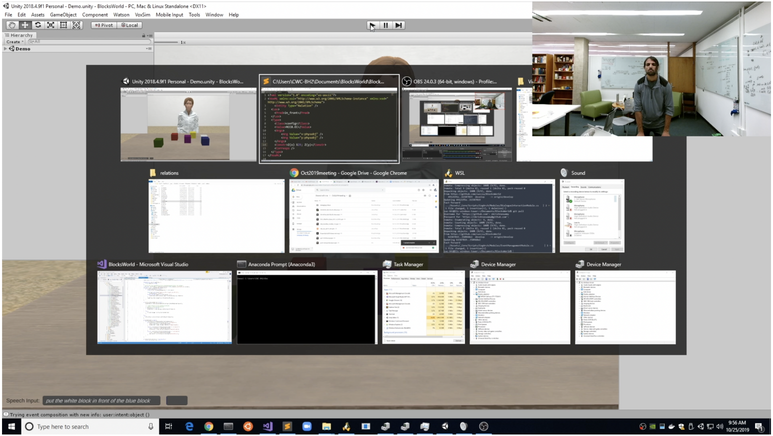
Spatial calculus
(e.g., RCC, TPCC, etc.)

$$
\left[\begin{array}{l}
\textbf{in\_front} \\
\text{TYPE} = \left[\begin{array}{l}
\text{CLASS} = \textbf{config} \\
\text{VALUE} = \boxed{RCC8.EC} \\
\text{ARGS} = \left[\begin{array}{l}
\text{A}_1 = \textbf{x:physobj} \\
\text{A}_2 = \textbf{y:physobj}
\end{array}\right] \\
\text{CONSTR} = Z(x) \boxed{>} Z(y)
\end{array}\right]
\end{array}\right]
$$

Mutable for frame of reference

VoxWorld Platform

VoxML operationalized



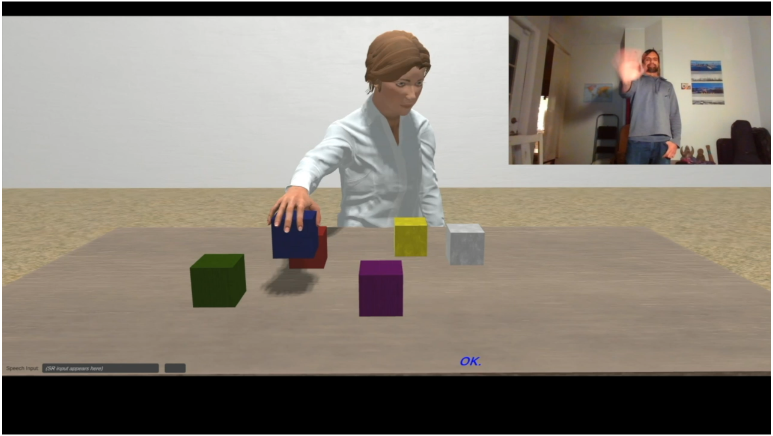
▸ Link

$$\left[\begin{array}{ll}\textbf{put} & \\ \text{BODY} = & \left[\begin{array}{lll}\text{E}_1 & = & grasp(x,y) \\ \text{E}_2 & = & while(hold(x,y) \wedge \neg at(y,z)) \\ & & \rightarrow move(x,y,z,\boxed{\mathbf{P()}},[loc(y),z,y]) \\ \text{E}_3 & = & if(at(y,z) \rightarrow ungrasp(x,y)\end{array}\right]\end{array}\right]$$

Pointer to optional function

(e.g., path planner)

VoxWorld Platform
VoxML operationalized



Link

What we have:

$$\begin{bmatrix} \textbf{put} & \\ \text{BODY} = \begin{bmatrix} \text{E}_1 = grasp(x,y) \\ \text{E}_2 = while(hold(x,y) \wedge \neg at(y,z)) \\ \quad \rightarrow move(x,y,z,\mathbf{P}(),[loc(y),z,y]) \\ \text{E}_3 = if(at(y,z) \rightarrow ungrasp(x,y) \end{bmatrix} \end{bmatrix}$$

$$\begin{bmatrix} \textbf{in\_front} & \\ \text{TYPE} = \begin{bmatrix} \text{CLASS} = \textbf{config} \\ \text{VALUE} = RCC8.EC \\ \text{ARGS} = \begin{bmatrix} \text{A}_1 = \textbf{x:physobj} \\ \text{A}_2 = \textbf{y:physobj} \end{bmatrix} \\ \text{CONSTR} = Z(x) > Z(y) \end{bmatrix} \end{bmatrix}$$

$$\begin{bmatrix} \textbf{block} & \\ \text{AFFORD\_STR} = \begin{bmatrix} \text{A}_1 = H_{[2]} \rightarrow [put(x,y,on([1]))] \\ \quad support([1],y) \\ \text{A}_2 = H_{[2]} \rightarrow [grasp(x,[1])] \\ \quad hold(x,[1]) \\ \text{A}_3 = H_{[2]} \rightarrow [lift(x,[1])] \\ \quad hold(x,[1]) \\ \text{A}_4 = H_{[2]} \rightarrow [ungrasp(x,[1])] \\ \quad release(x,[1]) \end{bmatrix} \end{bmatrix}$$

What we want:

$$
\left[\begin{array}{l}
\textbf{put} \\
\text{BODY} = \left[\begin{array}{l}
\mathrm{E}_1 = grasp(x,y) \\
\mathrm{E}_2 = while(hold(x,y) \wedge \neg at(y,z)) \\
\qquad \rightarrow move(x,y,z,\mathbf{P}(),[loc(y),z,y]) \\
\mathrm{E}_3 = if(at(y,z) \rightarrow ungrasp(x,y)
\end{array}\right]
\end{array}\right]
\left(
\left[\begin{array}{l}
\textbf{in\_front} \\
\text{TYPE} = \left[\begin{array}{l}
\text{CLASS} = \textbf{config} \\
\text{VALUE} = RCC8.EC \\
\text{ARGS} = \left[\begin{array}{l}
\mathrm{A}_1 = \textbf{x:physobj} \\
\mathrm{A}_2 = \textbf{y:physobj}
\end{array}\right] \\
\text{CONSTR} = Z(x) > Z(y)
\end{array}\right]
\end{array}\right]
\left(
\left[\begin{array}{l}
\textbf{block} \\
\text{AFFORD\_STR} = \left[\begin{array}{l}
\mathrm{A}_1 = H_{[2]} \rightarrow [put(x,y,on([1]))] \\
\qquad support([1],y) \\
\mathrm{A}_2 = H_{[2]} \rightarrow [grasp(x,[1])] \\
\qquad hold(x,[1]) \\
\mathrm{A}_3 = H_{[2]} \rightarrow [lift(x,[1])] \\
\qquad hold(x,[1]) \\
\mathrm{A}_4 = H_{[2]} \rightarrow [ungrasp(x,[1])] \\
\qquad release(x,[1])
\end{array}\right]
\end{array}\right]
\right)\right)
$$

Modeling Action Composition in VoxWorld

- Object Model: State-by-state characterization of an object as it changes or moves through time.
- Action Model: State-by-state characterization of an actor's motion through time.
- Event Model: Composition of the object model with the action model.

VoxWorld Platform

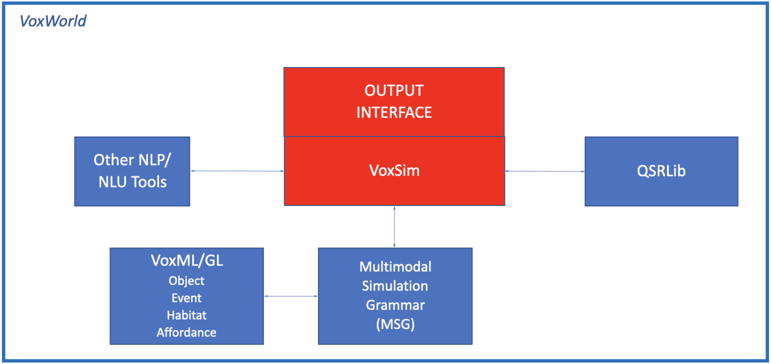


Figure: VoxWorld generic architecture

VoxWorld Platform

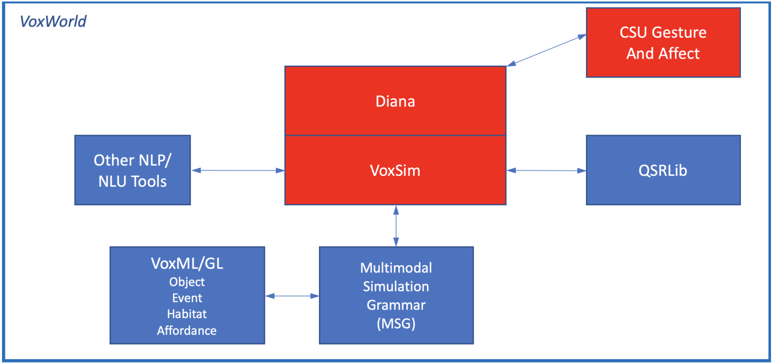


Figure: Agent implementation on top of VoxWorld

Diana Interactive Agent

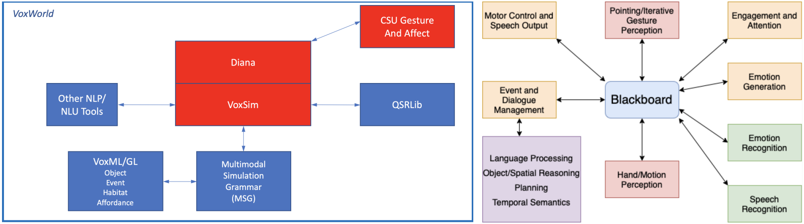
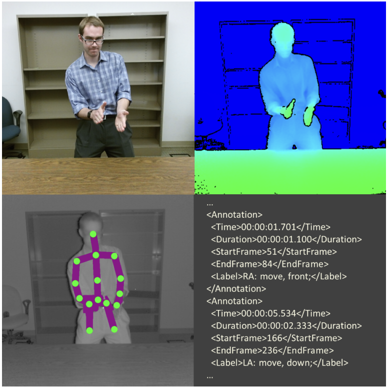


Figure: [L] Diana VoxWorld architecture; [R] Diana blackboard architecture

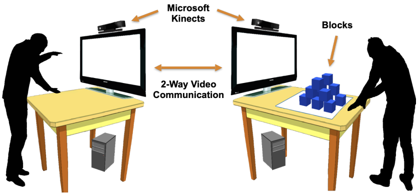
Gesture Recognition with CNNs

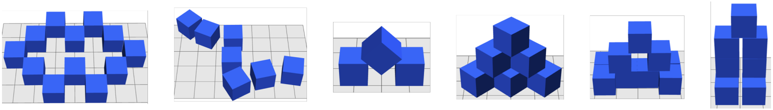
EGGNOG dataset

- How do people use gesture and speech together?
- EGGNOG: Elicited Giant Gallery of Naturally Occurring Gestures
- 60 Participants
- Over 8 hours of data
- 24,503 segmented and labeled movements



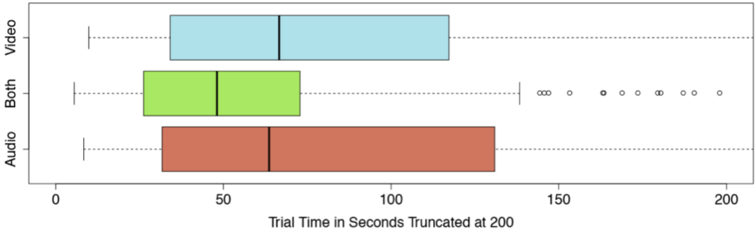
- Gesture only: Audio disabled, non-verbal communication only
- Speech only: Audio is enabled, video is disabled
- Speech and Gesture: Both audio and video are enabled





Gesture Recognition with CNNs
EGGNOG dataset

- Humans are surprisingly good at this!
  - With gestures alone, only 3 out of 200 trials failed
- When both modal channels are available people are much faster
  - A gesture is worth ~4-5 words



Gesture Recognition with CNNs

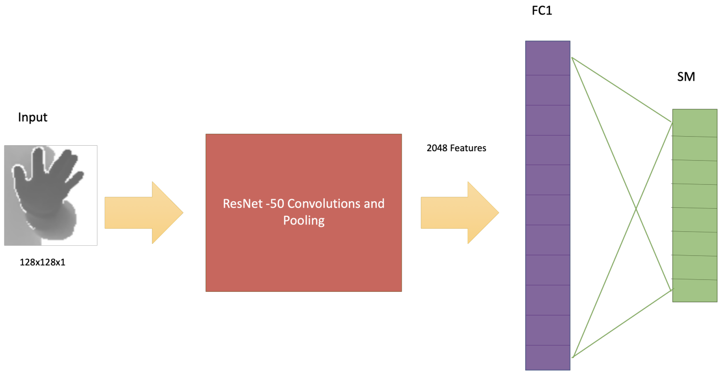


Figure: ResNet-style CNN for gesture recognition

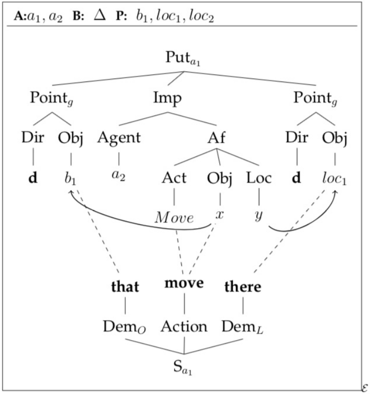
Unimodal Communication

Gesture only

Link

Language alone:

1. Human points to b_1
2. Diana points to b_1
3. b_1 established in common ground
4. Human points to b_2
5. Diana places b_1 on top of b_2



Unimodal Communication
Language only

Link

Language alone:

1. Human says "put a block next to the purple block"
2. b_1 — red block — introduced into common ground
3. Diana picks up white block$_{b_2}$ and puts it beside b_1
    - *a(block)* evaluates to random selection $\in \{b_1, b_2, b_3, \ldots\}$
    - (all except purple block)

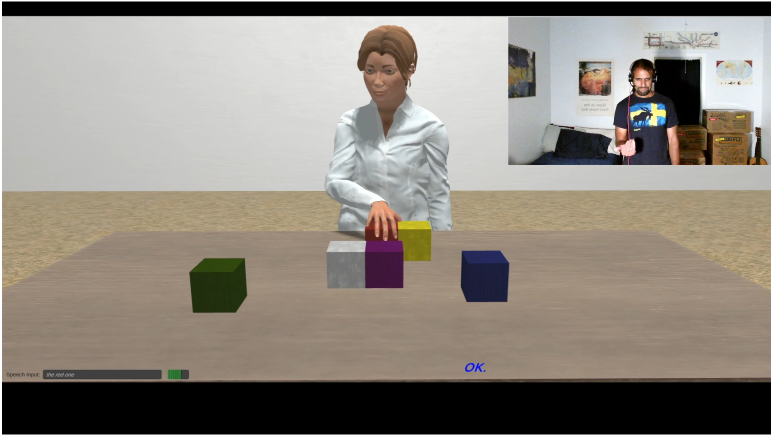
Multimodal Communication

Ensembles

Link

Mixing gesture and language:

1. Human says "the red one"
2. Diana points to b_1
3. b_1 — red block — established in common ground
4. Human makes *slide* gesture to right$_h$
5. Diana slides b_1 up against b_2
6. Implicit argument b_2 established in CG



Link

Bridging the same object across actions:

1. Human points to b_1
2. Diana points to b_1
3. b_1 established in common ground
4. Human makes *slide* gesture to right$_h$
5. Diana slides b_1 up against b_2
6. Implicit argument b_2 established in CG, b_1 still focused
7. Human makes *beckon* gesture (slide toward me$_h$)
8. Diana executes action over still-focused b_1

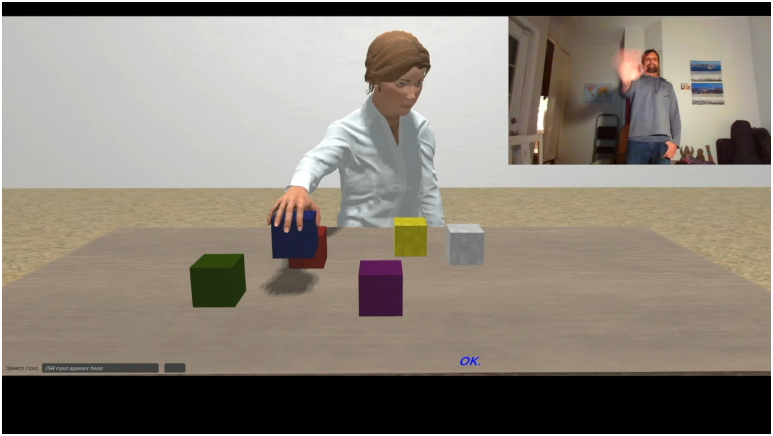
Link

Object coercion to location:

1. Human points at b_1
2. Diana points at b_1
3. b_1 established in common ground
4. Human says "put the yellow block *there*" + points
5. b_2 — yellow block — introduced into CG
6. b_1 coerced to location $l_1 := loc(b_1)$
7. l_1 introduced into CG

- *Potential future action: "put the blue block on it* — what should this do?

Correction and Clarification



Link

Correction and Clarification

- On a "correction" signal (e.g, *no*, *wait*, *stop*[1], etc.), agent should:
  1. Consume the *replacement* content following the correction signal
  2. Replace *only* the element(s) of common ground that also satisfy the semantics of the replacement content
  3. Reassign replacement content to equivalent place in common ground
  4. Continue execution

[1] See David Traum's talk in AREA-2 workshop!

Correction and Clarification

- Replacement content should accommodate multiple modalities and valencies (replace multiple types of CG information)
  - *"no, there"* (+deixis), *"stop, on the white one"*, *"no, wait, the red one"*, *"no, that one"* (+deixis), etc.
- Should **rewind** state monad, and replace where appropriate, keeping other content
- Result is an executable event with completely-specified semantics, but different at $\mathbf{S}_{k+1}$ than at $\mathbf{S}_k$

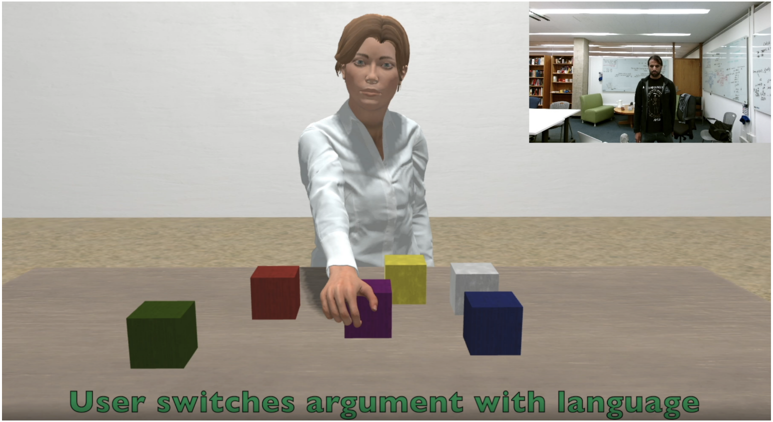
Interruption during Dialogue

Undoing Action on a Specific Block

Link

Interruption during Dialogue - Under the Hood

Correcting and Undoing Parameter Binding in Actions



Link

Interruption during Dialogue - Under the Hood

Correcting and Undoing Parameter Binding in Actions

$\lambda k.\overline{C_1}(\lambda n.\overline{C_2}(\lambda m.k(m\ n)))$

- $\lambda k_{Gib} \otimes k_{Telic}.k_{Gib} \otimes k_{Telic}(block)$
- $grab \subseteq \mathbf{sel}\ k_{Gib}$
- $\lambda k.k(grab) \Longrightarrow M, cg_1 \vDash grab(purple)$
- *"Wait, the yellow one."*
- **undo** $k = \lambda k.k(grab)$
- **Rewind** the state monad and **Reassign**:
- $\lambda k_{Gib} \otimes k_{Telic}.k_{Gib} \otimes k_{Telic}(block)$
- $grab \subseteq \mathbf{sel}\ k_{Gib}$
- $\lambda k.k(grab) \Longrightarrow$
- $M, cg_1 \vDash grab(yellow)$

- Situated grounding is particularly useful for transfer learning, because similar concepts often exist in similar situations (cf. analogical generalization, a la Forbus et al. (2017)).
  - e.g., "Build an X out of *these*," "Put *all those* in that X."
- Associate affordances with abstract properties—spheres roll, sphere-like entities probably do too.
- This informs the way you can talk about items (in real or virtual situations).
- Q: "What am I pointing at?" A: "I don't know, but it looks like [a container, something that rolls, etc.]"
- Similar objects have similar habitats/affordances.
- What happens when Diana encounters a new object?

- Exploit the correlations between habitats and affordances over known objects, and map those correspondences to novel objects
- Given: Object + A_1 + A_2 + ____ + A_4, predict A_3
- Goal: "Spheres roll. An apple is spherical. Apples probably roll."
- 17 distinct VoxML objects (~22 distinct affordance encodings):
  - e.g., $H_{[3]} = [\textsc{up} = \mathit{align}(\bar{Y}, \mathcal{E}_Y), \textsc{top} = \mathit{top}(+Y)]$, $H_{[3]} \rightarrow [\mathit{put}(x, \mathit{in}(\mathit{this}))]\mathit{contain}(\mathit{this}, x)$;
- Train 200-dimensional habitat or affordance embeddings using a Skip-Gram model;
- Represent objects as averaged habitat or affordance vectors.

- 2 architectures: 7-layer MLP and 4-layer CNN w/ 1D convolutions
- Evaluate against a ground truth of k-means clustered objects derived from human annotators

- Achieve ~80% accuracy with the predicted object clustering with the ground-truth object
  - ~40% of the time the predicted object *always* clusters with the ground truth in 5 randomized trials

Model	% predictions in correct cluster	% predictions always in correct cluster
MLP (Habitats)	78.82%	27.06%
MLP (Affordances)	**84.71%**	38.82%
CNN (Habitats)	78.82%	27.06%
CNN (Affordances)	81.18%	**40.00%**

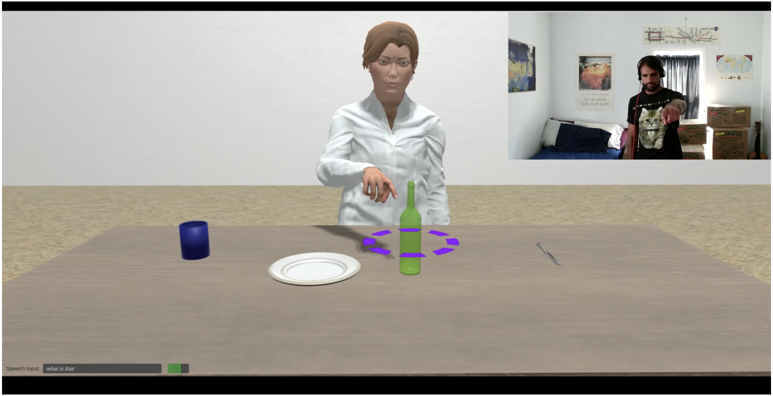
Tests on individual objects (plate):

Model	MLP-H	MLP-A	CNN-H	CNN-A
Predicted objects	book, cup, bowl, bottle	cup, bottle, apple	book	cup, bottle

- Habitat-based model typically better at capturing common behaviors (e.g., grasping), affordance-based model better at object-specific behaviors (e.g., rolling)

Learning with Affordances

Affordance Embeddings



Play!