Using POMDPs to Learn Language in a Spatial Reference Game

Suvir Mirchandani Levi Lian Benjamin Newman
{smirchan, levilian, blnewman}@stanford.edu

Abstract—**Much of early human language learning takes place in an unsupervised setting. In this work, we investigate how autonomous agents can use goal-oriented tasks in a spatial reference game to learn language. This problem is made difficult by the high dimensionality of the state and action spaces as well as the fact that it relates achieving one objective (i.e. reaching a goal) to achieving a secondary one (i.e. learning directional language). We formalize this problem as a Markov decision process (MDP) and partially observable Markov decision processes (POMDPs). We analyze the performance of the agent under different conditions using dynamic programming and online POMDP solution techniques. We perform and visualize simulations of the policies and real-time update of belief states. We observe that knowing the language can influence the time it takes to arrive at a goal state, and completely learning the language can be incentivized by explicitly optimizing for that task.**

I. Introduction

A. Background

Human language is at its a core a method for us to share collective experience and knowledge. Understanding language is therefore a social process that depends on speakers' reasoning about each other's intentions and goals [1]. In 1975, linguist Paul Grice formalized many of the rules that speakers tend to follow when cooperatively interacting with one another. These include maxims such as "make your utterances truthful"; "be informative and relevant"; and "avoid unnecessary prolixity and ambiguity." Grice's maxims have been experimentally observed through *reference games*, in which study participants have to identify an object picked out by another participant after hearing a word selected from a small vocabulary [2]. For these reference game scenarios, the maxims have also been computationally modeled in a Bayesian process where the agent hearing the utterance reasons about the speaker's beliefs of their own understanding [3].

Some prior work looks beyond single step reference games to multi-step interactive games. In these, multiple agents either try to find a certain unknown location or to compel each other to accomplish some task [2], [4]. These so-called *spatial reference games* provide a more challenging task as now agents have to model how their beliefs over each other's beliefs change as the game progresses. Vogel, Potts, and Jurafsky [5] formulated this problem as a decentralized-POMDP while Mordatch and Abbeel [4] used a deep neural network as a policy function.

We investigate a related but somewhat orthogonal question. In the aforementioned experiments, there were pre-defined messages that the listening and speaking agents could use and understand. In this project, we are interested in relaxing the constraint that both parties know the language being used to communicate *a priori*.

B. Related Work

Collaboration between humans and robots with natural language understanding systems is an active research area. While the traditional method is to learn from annotated text, grounding language in simulated environments has led to several novel approaches. Vogel and Jurafsky [6] derive the mapping from features of paths to the instruction language using reinforcement learning. This mapping helped an agent learn how to use natural-speech instructions to navigate around a map. In their paper, the state space combines world and linguistic features, representing both the physical positions and the interpretations of the human instructions.

While those methodologies focus on low-level commands (e.g., up, down, left, right) which directly map to movements, another approach focuses on finding the policy given a natural language instruction such as "Give me the water bottle." MacGlashan, Littman, Loftin, *et al.* [7] use reward and punishment for these training high-level tasks. Notably, rather than updating the belief state over possible actions to take, this approach induces a probability distribution over the set of possible tasks. The agent then plans a solution using an MDP planning algorithm and begins following the policy.

Conveying tasks through natural language provides an intuitive interface that does not require any technical expertise, but implementing such an interface requires methods for the agent to learn natural language commands grounded in the world, a difficult learning task. We therefore experiment with the interplay between a spatial goal and learning command meanings.

C. Problem Definition

In order to relax the constraint that both parties have prior knowledge of the language, we define a spatial reference game variant. In our game, there are two agents in a 2D rectangular room along with one goal location in the center of the room.

In our game, the agent that knows the location of the goal (the *speaker*) cannot move (and is not visible in the room), but it can speak to the agent that is searching for the goal (the *listener*). The listener can move but cannot speak. There are four utterances the speaker can use to direct the listener, which correspond to UP, DOWN, LEFT, and RIGHT. The listener can move in any direction from its current location.

We begin with the listener agent knowing the directions that each utterance corresponds to, along with its own location. We then remove its knowledge of its location, and finally, we take away its knowledge of what the utterances mean. Through this process, we analyze the response of an agent to directions and investigate its ability to learn the meanings of previously unknown instructions by the process of navigating to a goal.

II. Baseline MDP Formulation

A. Method

To start, we formalize this problem as a Markov Decision Process (MDP), where both the agent's location and directions are known to the agent. Later we will relax these assumptions. To formalize this MDP, we must define the necessary variables: the states $\mathcal{S}$, actions $\mathcal{A}$, transitions $\mathcal{T}$, and rewards $\mathcal{R}$.

- $\mathcal{S}$:
  - (x, y): the agent's x and y coordinates in the room.
  - σ: the command or instruction that the agent would receive from the speaker indicating the direction in which they should go.
  - status: whether or not the agent has reached the goal state.

  For implementation, we discretize the state space with 50 possible x-values and 40 possible y-values. There are 4 possible command values and 2 possible status values for a total of 16000 possible states.
- $\mathcal{A}$:
  - a:one of 24 equidistant angles that cover the range between $[-\pi, \pi)$. These represent the direction in which to travel. The agent moves a fixed step-size in every time-step.
- $\mathcal{T}(s, a, s')$: transitions are deterministic in this formulation; movement from one state to another occurs in the direction of the action. Each component of the state updates as follows:
  - $(x', y') = (x + \Delta p \cos(a), y + \Delta p \sin(a)$, where Δp is the constant step-size representing the constant velocity.
  - $\sigma' = \text{query\_speaker}(s')$. We define the function query_speaker as

    $$\text{query\_speaker}(s) = \text{round}(\arctan\left(\frac{g_y - s.y}{g_x - s.x}\right), \frac{\pi}{2})$$

    where (g_x, g_y) is the location of the goal position (which the *speaker* knows) and the round$(\cdot)$ function rounds the angle computed to the nearest $\pi/2$, representing the best direction to travel in. We associate the direction the speaker obtains with a command using the following table:

Command	Direction(s)
CMD1	$\pm\pi$
CMD2	$-\frac{\pi}{2}$
CMD3	0
CMD4	$\frac{\pi}{2}$

  - status' = $\mathbf{1}[(x', y') = (g_x, g_y)]$.

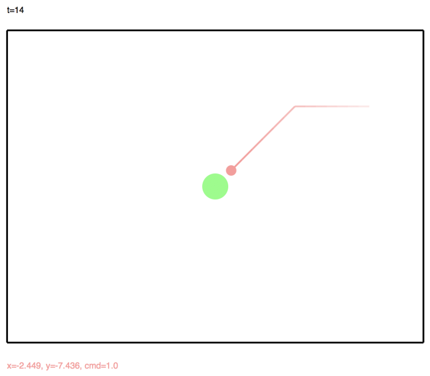

Fig. 1. An example result from value iteration. The agent reaches the goal in 14 timesteps. The green circle is the goal location and the red dot is the agent. The red line corresponds to path the agent took from its starting location.

  If the next state, s', is consistent with these updates, then the value of $\mathcal{T}$ is 1; otherwise it is 0.
- $\mathcal{R}(s, a)$: There are three reward sources we consider.
  - *Time Cost*: We penalize the agent with -0.1 for every step to incentivize taking actions to reach the goal quickly.
  - *Wall-contact Cost*: If an action takes the agent to a state where it makes contact with the edge of the room, there is a -1 reward.
  - *Reaching the Goal*: Arriving at the goal state gives a reward of $+100$.

    This means the final reward equation is:

    $$\begin{aligned}\mathcal{R}(s, a) = &- 0.1 - \mathbf{1}[\text{wall\_contact}(s.(x, y))] \\ &+ 100 \cdot \mathbf{1}[s.\text{status} = 1]\end{aligned}$$

B. Results

Because the action and state spaces are both discrete, we can apply value iteration to extract the optimal policy. In this formulation, the agent has perfect information about its position and the directions the commands map to, so we can use it as a baseline for our later models.

To start, we fixed the goal in the center of the 50×40 grid at position (-5, -10) and ran value iteration. We defined the agent to be "at" the goal when it was within two units of the goal position. With these conditions, we ran value iteration to extract the optimal policy. Simulating 100 trials led to a mean total reward of 99.097 (on average taking 10 steps to reach the goal position) and a standard deviation of 0.42.

Because the reward did not depend on following the command instructions at all, this MDP policy should be identical to the policy achieved by a similar MDP without the command in the state at all. In fact, running value iteration on that modified MDP with the same random seed obtains the same results over 100 trials: a mean total reward of 99.097 and a standard deviation of 0.42.

C. Metrics

To evaluate the success of our later experiments, we will compare the reward and number of timesteps computed given the same seed and number of simulations to those from the optimal policy above. This optimal policy gives us an idea of the best performance possible. In Table I, we summarize all the experimental results.

III. Experiments in Removing Information

Now we turn to the more interesting problem at hand: what changes when we remove some of the agent's knowledge of the state of the world? We investigate this broad question through two different experiments in which the agent no longer knows its own location and must determine how to get to the goal state based on the commands it receives. In the first experiment, we compare how quickly the agent arrives at the goal state when it knows the mappings between commands and directions, to the case where it does not have this knowledge. In the second experiment, we encourage the agent to learn the mapping from commands to directions by slightly altering the definition of the goal state. We then compare the agent's speed in reaching this goal state when it knows the command-direction mappings *a priori* to when it has to learn them from scratch.

In both of these experiments, we remove information from the agent by formulating the problem as a POMDP. The definition is very similar to the definition of the MDP above. Because now we are dealing with the command-direction mappings, we include this information in the state. The redefined state variable is:

$\mathcal{S}$:

- (x, y): the agent's x and y coordinates in the room. [Same as before]
- status: whether or not the agent has reached the goal state. [Same as before]
- σ_1, σ_2, σ_3, σ_4: all are angles in the range $[-\pi, \pi)$ indicating the agent's current guess of command-direction mapping. (σ_1 is the guess for CMD_1, etc.)

Because we changed the state, we have to modify the transition model as well.

$\mathcal{T}(s, a, s')$: The transition distribution is very similar to the original MDP's transitions. For the x, y, and status components, it is identical and it is still deterministic. To update the four command values, we query the speaker to determine what command we would get from the new position (x', y') and then use a temporal difference update (with $\eta = 0.3$) to push the agent's direction value for that command toward the direction that was taken, a. This is inspired from reinforcement learning. Formally, for the command components of s and s' we can define:

$$\text{next\_cmd} = \text{query\_speaker}(s')$$

$$t_{cmd}(s, a, s') = \begin{cases} s'.\sigma_1 = \eta(s.\sigma_1 - a) & \text{next\_cmd} = \text{CMD}_1 \\ s'.\sigma_2 = \eta(s.\sigma_2 - a) & \text{next\_cmd} = \text{CMD}_2 \\ s'.\sigma_3 = \eta(s.\sigma_3 - a) & \text{next\_cmd} = \text{CMD}_3 \\ s'.\sigma_4 = \eta(s.\sigma_4 - a) & \text{next\_cmd} = \text{CMD}_4 \end{cases}$$

When $t_{cmd}(s, a, s')$ is true, there is a valid transition (probability of 1) otherwise the transition probability is 0.

The reward model and actions remain the same.

Finally, because this is a POMDP, we need to define an observation distribution Ω that consists of the commands the agent receives at any given state. These are a deterministic function of the state and represent the direction from the agent's current position to the goal rounded to the nearest $\frac{\pi}{2}$. (This is the same as calling the query_speaker function defined for the transition distribution.)

$$\Omega(s, a) = \text{query\_speaker}(s)$$

Note that when we refer to the observations/commands received we use just the command name, but when we refer to a specific state s's mapping for the associated direction we use the syntax $s.\sigma_1$, $s.\sigma_2$, etc.

A. Belief State Representation

Because of the high dimensionality of our state space, instead of representing our belief over our current state explicitly, we sample 3000 points from the distribution and represent them as particles. We then update our belief over those points using particle filtering without replacement. When doing resampling, similar to Vogel, Bodoia, Potts, *et al.* [2], we add an additional weight factor to the belief update based on the command. We assign higher weights to particles whose command states (σ_1, σ_2, etc.) are closer (in angular distance) to the action we just took. More precisely, we calculate the angular distance between the direction the agent moved in and its direction values for each command. For the commands that we do not observe, we want this difference to be high, so we sum them. For the command that we do observe, we want this difference to be low, so we add π minus that distance. We normalize and compute the average of these values and use it to weight each particle. Formally, each particle has a weight:

$$W(o, a, s) = \frac{\pi - \text{dist}(s.o, a)}{4\pi} + \frac{\sum_{c \in \{\text{CMD}_1, \text{CMD}_2, \text{CMD}_3, \text{CMD}_4\}/o} \text{dist}(s.c, a)}{4\pi}$$

where $\text{dist}(\cdot, \cdot)$ is the absolute value of the angular distance between two angles, and $s.o$ is the σ associated with observation o.

We do not use the value for a directly. We perterb it slightly to avoid particle deprivation.

IV. Experiment 1: No Learning

In Experiment 1, we were interested in determining if the agent would be able to reach the goal state following commands it receives without knowing its position in the room, independent of knowing the directions the commands refer to. This experiment directly builds on the previous MDP one in that it keeps the original task—reaching the goal position—and just removes the agent's knowledge of its own location. There were two different conditions used in this experiment:

1) **KNOWN_COMMANDS**: In this condition, the POMDP is initialized with the commands mapped to the correct directions, and they are fixed. In other words, the value of the σ_1 is $-\pi$, σ_2 is $-\frac{\pi}{2}$, etc.
2) **UNKNOWN_COMMANDS**: In this condition, the POMDP is initialized with random values for each of the command mappings. The transition function includes the temporal difference update described above.

A. Hypothesis

Perhaps somewhat counterintuitively, we hypothesize that these two conditions will perform identically. Despite the difference in command mapping initialization, in both POMDPs, the goal state is a position in the rectangular room. This means the reward solely depends on the position and not on the command mappings. The agent will then try to arrive at this position by taking the action that brings its belief states closest to the goal *without any attention to the commands it receives*. Interestingly, even though the agent does not know its position, and the problem definition is slightly different from the MDP's, we expect the behaviour to be comprable to the MDP's. Intuitively, this is because each possible command specifies only one direction, so the agent will travel in one direction until it receives a new command and then change directions. We elaborate on this further in Section VI.

B. Methods

We use the POMDP implementations in `POMDPs.jl` [8]. Because the terminal state is so difficult to reach, when we tried using the `BasicPOMCP` solver to solve this modified POMDP, we were unable to reach convergence. The agent kept on getting stuck between two states, so we turned instead to a more powerful version of POMCP: POMCPOW [9]. POMCPOW originally was developed to extend Monte Carlo Tree Search and POMCP to continuous action and observation spaces. A major difference between these two algorithms is that POMCPOW requires a weighted particle filter. These weighted particles are necessary because POMCPOW simulates how the belief distribution (i.e. particles) will change over time compared to POMCP, which just simulates how the state (and action/observation history) changes. This is important in our case, because the belief distribution over our command mappings impacts what actions we take which then in turn determine how the state changes. Thus by simulating how the belief distribution evolves, we are considering many different potential ways to arrive at the correct command mapping. We use the upper confidence bound (UCB) strategy for choosing actions with a parameter value of 20.

C. Results

We simulated 100 trials for each condition. For Condition 1, the mean reward was 98.778 with a standard deviation of 0.832. The number of timesteps to solve had a mean of 11.32 and a standard deviation of 5.150. For Condition 2, the mean reward was 98.683 with a standard deviation of 0.764. The number of timesteps to solve had a mean of 12.07 and a standard deviation of 5.048. We performed a two-sample t-test that is robust to differing variances using the Welch-Satterthwaite procedure, finding that the two distributions are not statistically different ($p = 0.4498$).

Figure 2 shows a sample of both conditions solving the problem from a particular initial state.

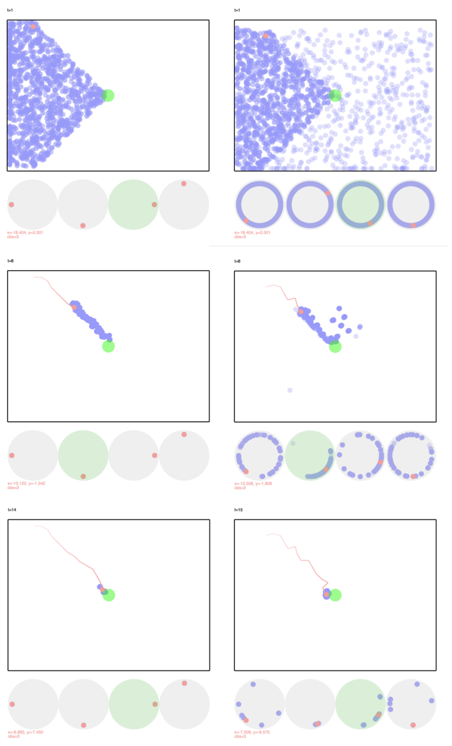

Fig. 2. A comparison of Condition 1 (left) and Condition 2 (right) at $t = 1$ (top), $t = 7$ (middle), and the last time step (bottom). The purple dots are particles in the state space which together represent the belief distribution. The four polar graphs below the room drawing render the command mapping direction for σ_1, σ_2, σ_3, and σ_4 (left to right).

V. Experiment 2: Learning the Language

We now discuss a more difficult and interesting problem—is it possible to learn which commands refer to which directions from scratch, and if so, how? To address this problem we make some more changes to our POMDP model.

First, we have to consider our goal state. In the previous experiments, we have assumed that the 'status' component of our state takes on a value of 1 when we arrive at the goal position. As pointed out in the Experiment 1 section, this is not suitable because there is no incentive to learn all of the correct direction mappings. Even worse, because the room is a rectangle, an agent will receive at most 2 distinct commands over the course of solving the POMDP (with an online solver such as POMCP) since it is grounded in a 2-dimensional space. In order to address this problem, we need to modify our terminal condition to include information about the command mappings we hold in our state. More precisely, we will say that we have reached the terminal state when we are at the goal position and each command ($sigma$) is within some threshold value, t, of the true direction that it should be mapped to. We used $t = \frac{\pi}{4}$. Mathematically:

$$\begin{aligned}\text{isTerminal}(s) = \mathbf{1}[(&\mid g_x - x \mid < 2) \wedge (\mid g_y - y \mid < 2)\\ &\wedge (\mid s.\sigma_1 + \pi \mid < t) \wedge (\mid s.\sigma_2 + \frac{\pi}{2} \mid < t)\\ &\wedge (\mid s.\sigma_3 \mid < t) \wedge (\mid s.\sigma_4 - \frac{\pi}{2} \mid < t)]\end{aligned}$$

While this goal state is the state the agent must be in to have learned the language, this is a very difficult state to reach. To quantify the difficulty, we again compare the results of solving this POMDP under the same two conditions as in Experiment 1:

1) **KNOWN_COMMANDS**: In this condition the POMDP is initialized with the commands mapped to the correct directions—or in other words, the value of the σ_1 is $-\pi$, σ_2 is $-\frac{\pi}{2}$, etc.
2) **UNKNOWN_COMMANDS**: In this condition, the POMDP was initialized with random values for each of the command mappings.

A. *Hypothesis*

Unlike in Experiment 1, here we hypothesize that if the commands are known, then the agent will reach the (modified) goal state much more quickly than if the commands are unknown. This is because if the commands are known, the problem collapses to finding the goal position (like in Experiment 1) whereas if the commands are initially unknown, the goal position as well as the command directions must be found. Arriving at the values of four additional angles in $[-\pi, \pi)$ takes more time.

B. *Methods*

We used POMCPOW as an online POMDP solution method for the same reasons we discuss in Section IV-B.

C. *Results*

We again performed 100 simulations for each condition, with an upper bound of 200 timesteps per trial. Samples that exceeded this bound were omitted from analysis. Under Condition 1, the mean reward was 98.782 wtih a standard deviation of 0.832. The number of timesteps to solve had a mean of 11.28 and a standard deviation of 5.152. Under Condition 2, the mean reward was 82.044 with a standard deviation of 30.619. The number of timesteps to solve had a mean of 98.36 and a standard deviation of 51.889. Here these distributions are statistically different ($p < 0.0001$, two sample T-test with unequal variance).

VI. Discussion

A. *Reaching the Goal State*

In both of these experiments, our hypotheses about the time it takes the agent to reach the goal state were confirmed. It appears that receiving commands indicating the direction of the goal does help the agent reach the goal, but they only help the agent learn the language when that is the explicitly set as the task. Why might this be the case?

In Experiment 1 we showed that the performance of an agent that knew the language *a priori* was the same compared to one that did not. In other words, knowing which command maps to which direction does not matter in reaching the goal state more quickly. This is a bit counterintuitive, but can be understood by considering that in this experiment, the agent only needs to localize itself to arrive at the goal position. Both conditions have access to the omniscient speaker, so they can use the command they received to determine what quadrant

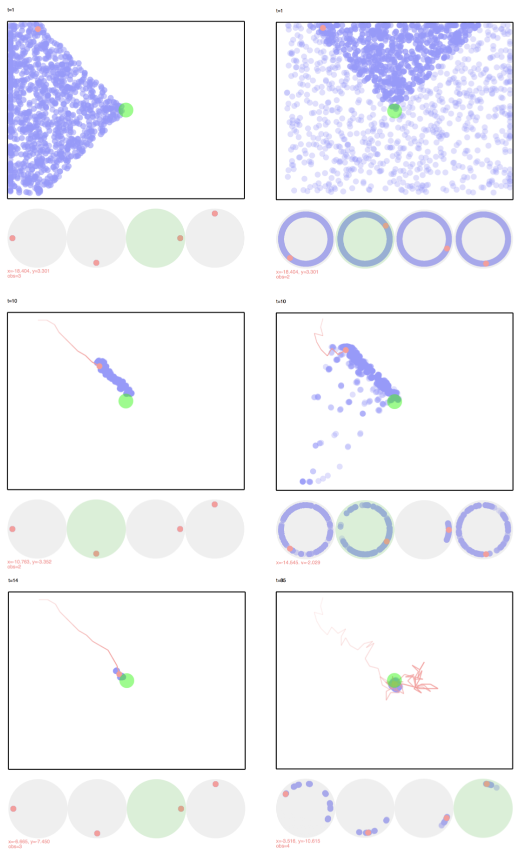
Fig. 3. A comparison of Condition 1 (left) and Condition 2 (right) at $t = 1$ (top), $t = 10$ (middle), and the last time step (bottom) for Experiment 2. The red line represents the agent's path and is more faint for earlier timesteps. Again the purple dots represent belief states and the intensity of the purple is the weight of that belief.

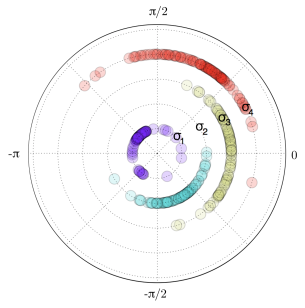

Fig. 4. Representation of the final command mappings of the 100 trials in condition 2 of Experiment 2. From the inside out we have the values of σ_1, σ_2, σ_3, σ_4.

TABLE I
EXPERIMENTAL RESULTS

Name	Reward Mean	Reward Std. Dev.
MDP with No Commands	99.907	0.42
MDP with Commands	99.907	0.42
Exp. 1 Cond. 1	98.778	0.832
Exp. 1 Cond. 2	98.683	0.764
Exp. 2 Cond. 1	98.782	0.832
Exp. 2 Cond. 2	82.044	30.619

of the rectangle they are in. Belief particles in locations that receive a different command are then discounted. This leads to a distinct triangle shape early on in the solving process (Figure 2, top row). Once the agent reaches the boundary between two command spaces, the triangle collapses to a single line of possibilities as other belief states not on that line can not be possible (Figure 2, middle row). Because the agent chooses an action based on the average of its belief states, it then heads toward the goal (Figure 2, bottom row). Nothing in this process depends on the agent's internal mapping from commands to directions. Only when we required the agent to have the correct direction mappings in the goal state did we observe that the condition where these mappings were randomly initialized performed worse. Interestingly, in Experiment 1, the average number of steps to reach the goal state in the POMDP case was very similar to the number it took to reach the goal state in the MDP scenario ($\sim$ 11 and 12 for the POMDPs and 10 for the MDP). This means that knowing the direction to the goal provides almost as much relevant information as knowing the actual current location.

B. Learning the Language

One of the major goals of this work was to investigate whether the mappings from commands to directions could be learned by working toward some, grounded, auxiliary goal (reaching a certain location). To this end we were somewhat successful. We can evaluate this success by looking at the values each command takes on in the terminal state in each experiment.

In Experiment 1, even though knowledge of the language does not impact how quickly the agent arrives at the goal position, the agent is still able to learn some of the correct command mappings. This is because the transition function takes into account the speaker's commands in the next state, and effectively pushes the values of the agent's internal command mappings (σ) toward the true directions of the commands. However, because any initial position will result in a maximum of two distinct commands observed, we can only expect the agent to learn two commands from a given starting position. This also a downside of using an online solver—if we could construct an optimal policy from each initial state, we might be able to extract the directions for each of the commands, but unfortunately the state space is too large and continuous to do so.

While learning the meaning of two commands is evidence of some success, it is very difficult to have the agent learn the meaning of all four commands under this framework. In Experiment 1 there is nothing compelling the agent to do anything but head straight for the goal. There are multiple ways to alter the problem to try and incentivize the agent to take actions that do not move it closer to the goal—such as placing the agent in a maze for instance—but we chose to modify the goal state which led to Experiment 2.

In Experiment 2, we can see that the agent first heads straight for the goal state and then seems to randomly wander back and forth around the goal position (Figure 3, bottom row). At this point, the agent has localized its belief states in the position space to its exact location. Then it seems to focus on arriving at the correct command mappings. To do this it has to observe each command multiple times, which is easier to do when the agent is closer to the goal state. The wandering takes the agent all around the goal position (particularly focusing on the boundaries between commands), so it is able to use the different angles to improve its predictions for the correct command mappings. We can see this in how the agent's path (the jagged red line) seems concentrated at the boundaries between commands 2 (down) and 3 (right) and 3 and 4 (up) in the bottom right image in Figure 3. This is a difficult problem, so it is quite impressive that only 8 of the 100 trials took more than the upper limit of 200 steps to converge at the correct command mappings (Figure 4). The POMDP solver seems to break down the problem into first arriving at the goal position (localizing self) and then moving around it (determining the rest of command mappings). This is reminiscent of how we as humans might achieve the task if we were in the agent's position.

VII. FUTURE WORK

We propose three different possible extensions of this work. First, in current experiments, the agent has no mechanism for asking clarifying questions to improve its knowledge of the commands. However, as Kollar, Tellex, Roy, *et al.* [10] have proposed, agents learn best when dynamically engaging with the human teacher based on a cost function over the possible actions and the parts of the command. Woodward and Wood [11] even frame the agent learning interactively from a human teacher problem as an interactive partially observable Markov decision process (I-POMDP), an extension of the POMDP to the multi-agent setting. This interactivity of language learning should be explored as future research. Finally, in Vogel, Bodoia, Potts, *et al.* [2]'s original paper, some of this interactivity was baked into the POMDP model. Modifying the query_speaker routine to take into account the speaker's belief over how much the agent has learned would be another interesting path to explore how Gricean pragmatics might play into language learning.

VIII. CONCLUSION

Much of human language has physical grounding and we learn language by hearing what people say when we interact with the world. In this project, we focused on language learning in this sense, attempting to teach an agent directional language. We formulated this language learning problem as a POMDP and observed modest success in teaching the agent what certain commands mean while the agent was attempting to achieve a secondary goal. Only once the goal was altered to explicitly require the agent to learn the entire language did complete understanding emerge.

IX. Contribution of Group Members

Discussions in the group as a whole were necessary for formulation of the problem and experiments we performed. Ben's linguistics background led him to suggest the field initally, and helped provide contextualization for the problem space. He also programmed the MDP and value iteration part of the project, and worked on the belief update function for the POMDP part.

Levi worked on modifying the Roomba environment to be suitable for this heavily modified problem, and also led the literature review section and statistical analysis.

Suvir's interest in visualization was useful for creating the UI to illustrate solvers and policies in real-time (modified from the Roomba problem) along with creating the data visualizations. He was responsible for the remaining POMDP coding along with testing out different solvers.

References

[1] M. C. Frank and N. D. Goodman, "Predicting pragmatic reasoning in language games," *Science*, vol. 336, no. 6084, pp. 998–998, 2012.

[2] A. Vogel, M. Bodoia, C. Potts, and D. Jurafsky, "Emergence of gricean maxims from multi-agent decision theory," in *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, 2013, pp. 1072–1081.

[3] N. D. Goodman and M. C. Frank, "Pragmatic language interpretation as probabilistic inference," *Trends in cognitive sciences*, vol. 20, no. 11, pp. 818–829, 2016.

[4] I. Mordatch and P. Abbeel, "Emergence of grounded compositional language in multi-agent populations," *arXiv preprint arXiv:1703.04908*, 2017.

[5] A. Vogel, C. Potts, and D. Jurafsky, "Implicatures and nested beliefs in approximate decentralized-pomdps," in *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, vol. 2, 2013, pp. 74–80.

[6] A. Vogel and D. Jurafsky, "Learning to follow navigational directions," in *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, Association for Computational Linguistics, 2010, pp. 806–814.

[7] J. MacGlashan, M. Littman, R. Loftin, B. Peng, D. Roberts, and M. E. Taylor, "Training an agent to ground commands with reward and punishment," in *Proceedings of the AAAI Machine Learning for Interactive Systems Workshop*, 2014.

[8] M. Egorov, Z. N. Sunberg, E. Balaban, T. A. Wheeler, J. K. Gupta, and M. J. Kochenderfer, "POMDPs.jl: A framework for sequential decision making under uncertainty," *Journal of Machine Learning Research*, vol. 18, no. 26, pp. 1–5, 2017. [Online]. Available: http://jmlr.org/papers/v18/16-300.html.

[9] Z. Sunberg and M. Kochenderfer, "Pomcpow: An online algorithm for pomdps with continuous state, action, and observation spaces," *arXiv preprint arXiv:1709.06196*, 2017.

[10] T. Kollar, S. Tellex, D. Roy, and N. Roy, "Grounding verbs of motion in natural language commands to robots," in *Experimental robotics*, Springer, 2014, pp. 31–47.

[11] M. P. Woodward and R. J. Wood, "Learning from humans as an i-pomdp," *arXiv preprint arXiv:1204.0274*, 2012.