Progress Report: Social Feedback for Robotic Collaboration

Emily Wu, Brown University

February 22, 2016

1 Introduction

In my proposal, I introduced a project to enable robots to interact socially with humans in human robot collaboration tasks. In order to establish this social feedback loop, robots must be able to both perceive and convey their mental state to the human partner. Presented below is what I have accomplished so far.

2 Implementing the Domain

For this project, I built a domain describing an object handover interaction on the BURLAP framework. In order to build a domain, the states, actions, observations, reward function, and their interactions must be described within the framework. I have implemented three different variants of these domains, and begun evaluation on them in a simulated environment.

2.1 Base implementation

In the basic implementation, we define the states, actions, transition function, reward function, and observations and observation models as follows:

States are given by a tuple of $s = \langle \omega, \mathcal{O}, \hat{b}_H \rangle$. The object the human wishes the robot to handover is w, the set of all objects and their locations and names are given as $\mathcal{O}$, and the robot's belief about which object the human believes they will hand over is represented by a distribution $\hat{b}_H$ over $\mathcal{O}$.

The **actions** available for the robot to take are to `WAIT` (take no action) `PICK(x)` some object x, `ASK(x)` about some object(s) x, `POINT(x)` to some object, or `LOOK(x)` at some object x.

The **transition function** describes how state changes in response to the robot's actions. The desired object ω only changes once the robot has `PICK`ed the correct object. Similarly, $\mathcal{O}$ only changes if an object was picked up. $\hat{b}_H$, the belief of the human about which object the robot hands over, changes with every action the robot takes. Specifically, $\hat{b}_H$ is updated in a Bayesian manner as if the human had observed the robot's action. Specifically:

$$b_{Ht}(\omega_t) = p(a_t|\omega_t) \sum_{\omega_{t-1}} p(\omega_t|\omega_{t-1}) b_{Ht-1}(\omega_{t-1})$$

$p(a_t|\omega_t)$ gives the probability of observing action a_t (the robot's last action) given an object ω_t, which is the object the human believes the robot is trying to communicate. this directly mirrors the observation functions described below.

The **observations** that the robot can make of its environment (i.e., the human's actions) are speech and gesture made by humans. They are represented by a tuple of $\langle l, g \rangle$, language and gesture. Speech is given by a NLP provided by Google, while gesture is provided by a Kinect.

The **observation function** describes how observations are produced by the current state. In a POMDP, the current state is hidden, so it is from these observations that the agent (robot) must determine the current state. Our observations pertain mostly to the desired object ω. The details of the observation function are unchanged from the proposal, but I use the same models to update the state variable $\hat{b}_H$, except from the perspective receiving observations from the robot's actions.

The language model is as follows:

$$p(l|s) = p(l|\omega) = \prod_{\text{word} \in l} \frac{\text{number of instances of word in } \omega\text{'s vocabulary}}{\text{total number of words in } \omega\text{'s vocabulary}}$$

The gesture model assumes that the participant picks a point to gesture at by sampling from a Gaussian distribution centered at the object they are pointing at. To get the probability of a given gesture, we return the density of a Gaussian distribution centered at 0 at the angle defined by the vector from the participant's shoulder to the object and the vector from the participant's shoulder to their hand, which we call θ_g. We choose a hand tuned variance v.

$$p(g|s) = \mathcal{N}(\theta_g|0, v)$$

We assume language and gesture are conditionally independent given the state, so our total expression for the observation function is

$$p(o|s) = p(g|s)p(l|s)$$

The **reward function** describes how the robot receives reward according to its actions and the hidden states. This essentially incentives and disincentivizes certain behaviors. In this domain, we provide a large positive reward for `PICK`ing the correct object, and a large negative reward for `PICK`ing the incorrect object. In addition, several smaller negative rewards are given for taking actions such as `ASK`ing the user questions or as a penaty for "annoying" the human. In addition, we give the robot a small reward for how closely $\hat{b}_H$ matches the distribution representing the robot's belief about which object the human desires, which we label b_R.

2.2 Incremental Picks

On top of the base implementation, I have experimented with variations that improve the robot's ability to communicate. The first of these variations is the inclusion of an incremental pick action. In this variation, the robot must deliberately choose the pick action until it observes the pick is complete in order to receive the positive (or negative) reward. This allows the robot to continue taking in information from the human as it is picking, as attempting to pick up the object is an indisputable sign of the robot's internal state, allowing the human to correct the robot if the robot is picking the incorrect object.

This experiment is running in simulation, where the desired effect of the robot canceling its pick if additional correcting information is observed. I have not yet run this domain with real robots as solvers (described later) for the POMDP behave poorly with regards to the delayed reward. This suggests approach that abstract out the actions the robot can take to allow for higher level planning, which should be more robust to delayed reward. In the coming semester, I will investigate Abstract MDPs (AMDPs) as a solution to this problem.

2.3 Alternative Reward Function

The reward function of the domain described above is incompatible with the POSS solver used to plan actions (see below), so an alternative had to be devised.

In the new version of the domain, we change the state representation from $\langle \omega, \mathcal{O}, \hat{b}_H \rangle$ to $\langle \omega_R, \omega_H, \mathcal{O} \rangle$. In this new state variable, ω_R is the same as the old ω: the object the human desires. The new variable ω_H represents the object the robot has communicated to the human through its actions. This is a hidden variable that the robot must maintain an estimate over, as it is not certain of the interpretation of its actions. This is a parallel interpretation of $\hat{b}_H$. The transition function was adjusted such that the underlying distribution over ω_H matched $\hat{b}_H$. Actions, observations and the observation function remained unchanged. On top of the existing reward function defined above, we also add an addition reward that operates over ω_R and ω_H .

$$R(\langle \omega_R, \omega_H \rangle) = \begin{cases} 3 & \text{if } \omega_H == \omega_R \\ 0 & \text{otherwise} \end{cases}$$

The reward returned by this function is added to the reward defined in the base domain.

This reward function is designed to offset the cost of taking more expensive actions: if you have correctly communicated the object you believe the human has to the human, the cost of communicative actions is offset. As a result, running the domain in simulation results in more communicative actions which reflect which object the robot believes the human desires.

However, mathematical analysis demonstrates that this reward function will not perfectly accomplish the behavior we desire. In rewarding ω_H and ω_R to be the same, we are essentially asking the robot to maximize the following expression:

$$\underset{\hat{b}_H}{\operatorname{argmax}}\, p(\omega_R == \omega_H) = \underset{\hat{b}_H}{\operatorname{argmax}} \sum_{\omega \in \mathcal{O}} \hat{b}_H(\omega) * b_R(\omega)$$

The resulting $\hat{b}_H$ has the entirety of the probability mass on the object with greatest probability in b_R, so the robot will express full certainty on the object with the greatest probability in b_R. This is not the behavior we would like the robot to present—rather, we would prefer if the robot could express the uncertainty about which objects the human desires as represented by b_R, which requires $\hat{b}_H$ to match b_R as closely as possible.

Because this version of the domain does not produce the behavior we want, the next step is to address the original incompatibility of the reward function to work with the POSS solver. This line of inquiry was also useful, and showed that rewarding the robot to offset the cost of communication is a good means of producing reliable and consistent communication.

3 Performance and Solvers

A POMDP domain is a problem formulation; solvers are programs that use the problem formulation to derive a solution. In this case, a solution is an action that should be executed in response to an inputted state. Solving a POMDP is a computationally expensive task that can take anywhere from a few seconds to several minutes to complete. This is not ideal for human-robot interaction, as, in the meantime, the human participant is waiting or giving the robot additional commands that it cannot incorporate as it plans its next action. In previous work, this was addressed by precomputing the correct action ahead of time and using nearest-neighbor to choose actions at the time of interaction. This approach provided near instantaneous results, but had the downside of imperfect performance at completely novel states as well as a loss of flexibility, as the policy would have to be recomputed if anything about the domain were to change.

Since then, we have addressed the problem by using a new solver called Partially Observable Sparse Sampling (POSS) which provides much faster performance; it averages 2 seconds per action for 4 objects or 5 seconds per action for 6 objects real time, with room for optimization. This allows us to change the objects on the table freely without having to recalculate anything.

However, 5 seconds is still a long latency for interactive tasks, especially if the robot is no longer observing the human's actions and is fully concentrated on planning its next one. For this reason, I have made a multithreaded version of this program which allows the robot to continue observing and updating its state as it plans. Though untested with human participants so far, the hope is that it will mitigate the latency while the robot plans and maintain a more accurate robot state.

4 Theoretical Formulation

So far the description of the domain has been specific to its actual implementation. We would like to verify its correctness by providing a full theoretical formulation that can be simplified to an achievable implementation.

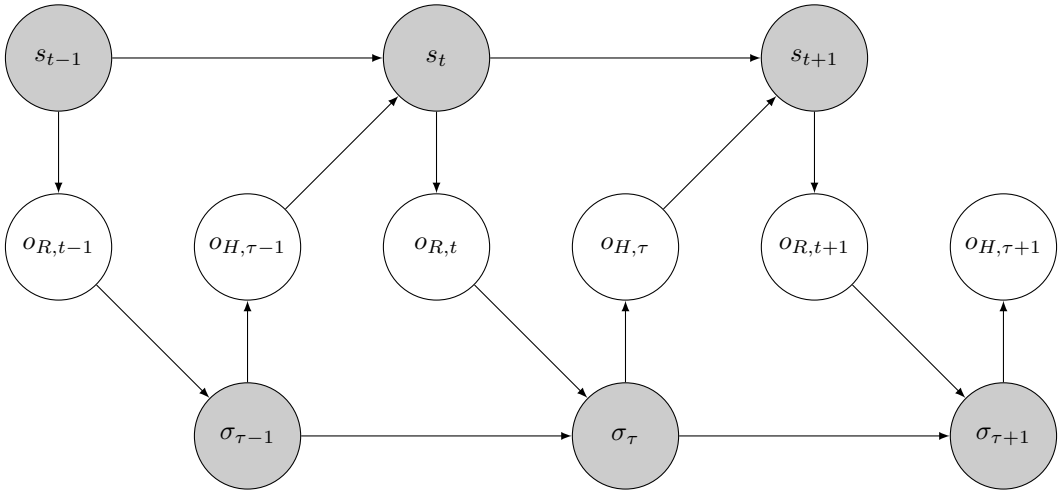
In particular, we would like a full description of the system that describes how the human updates its belief in response to the robot's actions, as well as how the robot should update its belief about the human's interpretation of the robot's actions.

4.1 Variables

We define two agents, the human and the robot. Each agent has their own state, composed of a tuple $s = \langle i, b, B \rangle$ for the robot and $\sigma = \langle \iota, \beta, \mathcal{B} \rangle$ for the human. The robot's i is the object it will hand over to the human whereas the human's ι is the object the human desires. Both i and ι are hidden from their counterart, so the robot has a state variable $b(\iota)$ that is a distribution over ι, representing its belief about which object the human desires. Likewise, the human maintains a state variable $\beta(i)$ over the human's object i that tracks the human's belief about which object the robot will hand over. Once again, both b and β are hidden from their counterparts, so the robot maintains a distribution over β called $B(\beta(i))$ and the human maintains a distribution over bs called $\mathcal{B}(b(\iota))$.

In addition, we define an action a_R to be an action performed by the robot and observed by human, and a_H to be an acton perfomed by the human and observed by the robot. Because these action are also observble, we will also refer to actions takn by the robot as o_R and actions taken by the human o_H in contexts where they are treated as observations.

All of the variables mentioned above can be parameterized by time steps. For the robot, actions, observatons, and state variables will be parameterized by a time t. For the human, we will use time varible τ, where τ is defined a half step after t, i.e., $\tau = t + 0.5$. This is illustrated in the diagram below.



Notice that when $o_{R,t}$ is treated as an action, its time step is changed to $a_{R,\tau}$. Similarly, when $o_{H,\tau}$ is treated as an action, we call it $a_{H,t+1}$.

We would like an expression that gives us $p(s_t|a_{H,1:t})$. Breaking the state into its three components, we must define update expressions for $p(i_t|a_{H,1:t})$, the object level update; $p(b_t|a_{H,1:t})$, the categorical level update; and $p(B_t|a_{H,1:t})$, the dirichlet level update.

4.2 Item Level Update

The question of how "which object does the robot hand to the human given the human's actions" is nonsensical from the perspective of the robot, since this is exactly what we are trying to decide by solving our POMDP. However, since the object the robot will hand over is unknown to the human, it is useful to define a transition function for the human to use as an estimate to update its belief about the robot's i, $\beta(i)$. This function is given below:

$$\begin{aligned} p(i_t|a_{H,1:t}) &= \sum_{i_{t-1}} p(i_t, i_{t-1}|a_{H,1:t}) \\ &= \sum_{i_{t-1}} p(i_t|i_{t-1}, a_{H,1:t}) p(i_{t-1}|a_{H,1:t}) \\ &= \sum_{i_{t-1}} p(i_t|i_{t-1}, a_{H,t}) p(i_{t-1}|a_{H,1:t-1}) \end{aligned}$$

4.3 Categorical Level Update

The update to b and β is a standard hidden markov model update over objects i where the observations are actions a.

$$b_{Rt}(i_t) = p(i_t|a_{1:t}) = p(a_t|i_t) \sum_{i_{t-1}} p(i_t|i_{t-1}) p(i_{t-1}|a_{1:t-1}) \tag{1}$$

$$= p(a_t|i_t) \sum_{i_{t-1}} p(i_t|i_{t-1}) b_{Rt-1}(i_{t-1}) \tag{2}$$

This update can be written as a matrix multiplication:

4.4 Dirichlet Level Update

Now that we've defined how b_R is updated, we can define B_H in terms of b_R's update.