Social Feedback for Robotic Collaboration

Emily Wu, Brown University

April 7, 2016

Contents

1 Abstract

2 Introduction

Collaboration is a process that relies heavily on communication for its success. When humans collaborate, this communication is both obvious and implicit—we not only instruct and request aid from each other, but we also tacitly monitor our partners for signs of approval and understanding while producing these signals ourselves. However, in robotics, many of these vital components of successful communication are missing or lost. The absence of these signals likely account for numerous failures in human-robot collaborative tasks. In order to give robots these missing communicative skills, we employ *social feedback* signals to provide human-like communication to both inform the human participant of the robot's state and request information from the participant.

Feedback refers to the responses that are received by an agent when it takes some action. These feedback responses inform the agent about the success or failure of its actions. *Social feedback* therefore refers to social signals that convey this information. These can be explicit signals, such as "I want this object", or implicit signals, like a perplexed expression. When humans interact with another human, they use this feedback to improve the flow and clarity of the collaboration, which in turn improves its success. For my thesis work, I have developed a framework that describes how robotic agent should likewise generate social feedback for its human partner in human-collaborative tasks, and show that it improves the speed and accuracy of human-robot interaction.

In this work, we address the object delivery task. In this task, a set of objects are laid out on a table within the robot's reach. A human participant requests an object from the robot using speech and gesture (pointing). The robot must interpret the human's speech and gesture and deliver the requested object to the human. The task then repeats with the remaining objects on the table. This task is achievable without social feedback; the robot need only wait until enough information is given and then deliver the correct object. However, we will show that by adding social feedback actions, such as asking questions, looking at objects, and pointing at objects, we will achieve better accuracy and speed as well as improved user experience.

We will solve this problem by formulating it as a Partially Observable Markov Decision Process (POMDP), which will allow us to dynamically and flexibly determine how to choose social feedback actions. First, we will describe a two agent model and use this to motivate the construction of a POMDP. We will next discuss how we solved this POMDP and other measures we took to allow our system to respond dynamically and fluidly.

To evaluate our approach, Stop. Hammertime.

3 Related Work

4 Technical Approach

4.1 POMDP Overview

Partially Observable Markov Decision Processes (POMDPs)**citation** are used to model Markov Decision Processes**citation**where the true state is not known. Instead, the agent receives observations that are generated by the true state, and must infer what the true state is from the observations. Thus, the agent maintains a belief over true states which is updated as it receives new observations. The agent must then use this belief state to determine which action to take to maximize its expected value of the reward over time. This splits the POMDP into two main components, a state estimator and a policy generator.

State Estimator The state estimator is given an initial belief and uses Bayesian mathematics to update its belief over time. In order to perform this update, the state estimator has a model of how the hidden states emit observations as well as how the state evolves in response to actions performed by the robot. The first model is described by the observation function and the second by the transition function.

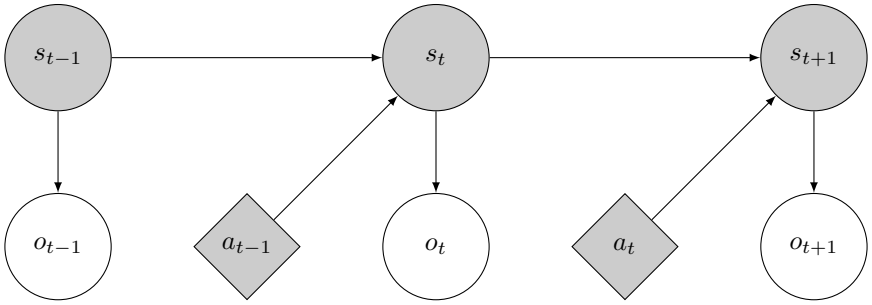
Policy Generator The policy generator is the means by which actions are chosen by the agent. Specifically, the policy generator should choose actions that maximize the agent's expected reward over time. The reward for taking a particular action in a state is given by a reward function. These state-action pairs are called policies, and are generated by a POMDP Solver. Generally, POMDPs are solved by converting to their equivalent belief-MDP, which is an MDP where the state is distribution over the POMDP's hidden states, a belief. In subsequent sections we will refer to this belief over states as b.

Formally, a POMDP is defined as a tuple $\{S, A, T, R, \Omega, O\}$, where :

- $s \in S$ is a hidden state.

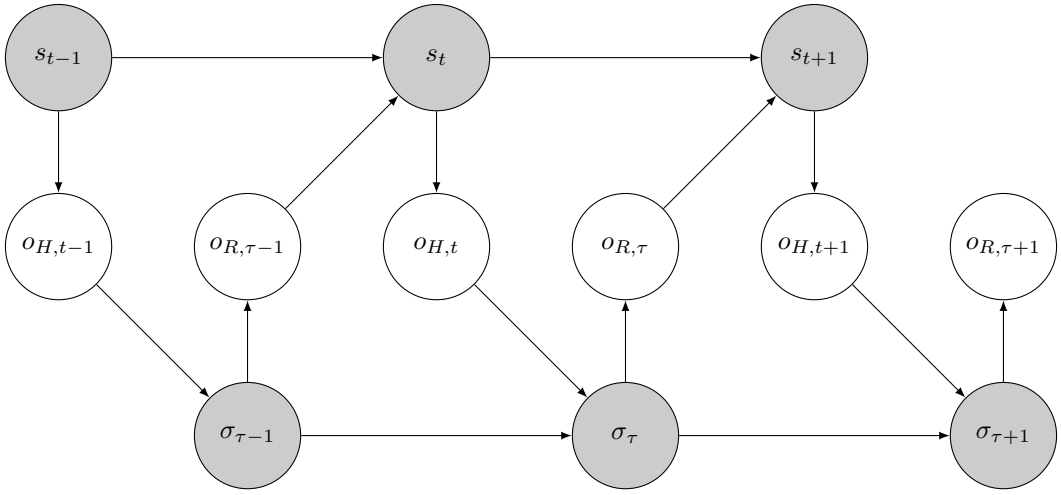
- $a \in A$ is an action that an action takes.
- $T : S \times A \times S \to \mathcal{R}$ is a transition function that returns returns the value $p(s'|s, a)$, the probability of transitioning to state $s' \in S$ from state $s \in S$ when taking action $a \in A$.
- $R : S \times A \to \mathcal{R}$ is a reward function that returns the reward for taking action $a \in A$ in state $s \in S$.
- $o \in \Omega$ is an observation.
- $O : S \times A \times O \to \mathcal{R}$ is an observation function that returns the value of $p(o|s, a)$, the probability of observing observation $o \in O$ when action $a \in A$ is taken from state $s \in S$.

A typical bayesian network for a POMDP is as follows.



4.2 Model Description

To model this human-robot interactive task, we will use a POMDP. To motivate the construction of our model, consider the following two-agent Bayesian network.



In the above model, we represent human states s_t, robot states σ_τ, and observations generated by the robot $o_{R,t}$ and by the human $o_{H,t}$. Observe that the structure of this model resembles two POMDPs combined together at their actions and observations. The lower POMDP is the POMDP from the human's perspective: the robot has some hidden states s_t, which the human observes by means of $o_{R,t}$. The human takes actions $o_{H,\tau-1}$ to influence the robot's state s_t. The upper POMDP is likewise the a POMDP that models the robot's process: the human's state σ_τ is hidden from the robot, and the robot must infer it from observations $o_{H,t}$. When the robot takes action $o_{R,t}$, it affects the human's state σ_τ. Crucially, each agent treats the other agent's as an action that influences their belief about the other agent's hidden state. Thus, the human's actions affect the robot's belief about the human's state, which is what we call b as defined above. Importantly, the reverse is also true: the robot's actions affect the human's belief about what the robot's hidden state is. We will call the human's belief over the robot's hidden state β.

In the following section we will use this dual structure to inform the construction of our POMDP as applied to our object delivery domain.

4.3 POMDP Definition

We define our object-delivery POMDP as a tuple $\{S, A, T, R, \Omega, O\}$:

- Each $s \in S$ is a tuple of $\langle \iota, \beta, \mathcal{I} \rangle$
  - $\mathcal{I}$ is the set of all objects that the robot can deliver. Each object is parameterized by a name, a unigram vocabulary, and a position; for example: a red bowl would be represented ⟨redBowl, [red, red, bowl, bowl, plastic], (1.0, 2.0, 0.0)⟩. We assume the set of all objects is known.
  - $\iota \in \mathcal{I}$ is the object the human desires. This is a hidden variable.
  - β is a distribution over the robot's hidden states, as defined above. In this domain, the robot state σ represents which object the robot will hand the human, or which object the robot believes the human wants.
- The set of actions A consists of both social and non-social actions. Non-social actions are `pick(i)` (picking up and delivering an object i) and `wait`. Social actions are `point(x)`, pointing at a location x; `look(x)`, looking at a location x, `say(p)`, informing the human that the robot believes the desired object has property p. p is an element of some object's vocabulary.
- $T(s, a, s') = p(s'|s, a)$. We make the assumption that the human's desired object ι does not change unless the robot picks up object ι. The set of all objects $\mathcal{I}$ transistions to $\mathcal{I} \setminus \{i\}$ when the robot chooses action `pick(i)`. For each action in A, we define a "reverse observation function" that describes our assumptions about how the human imagines the robot generates actions (which the human sees as observations) given a hidden robot state σ. This is gone into further detail in section 4.3.2.
- $R(s, a)$ takes as input a state $s \in S$ and an action $a \in A$. In this domain, we incentivize our robot to pick the correct object by giving it a +10 reward if it delivers the correct object and a -50 reward for picking the incorrect object. We also give negative rewards for taking varioius actions: `wait` receives a -1 reward (to incentivize the robot to finish the task quickly); `look(x)` receives a reward of -2; `say(p)` receives a reward of -3; `point(x)` receives a reward of -4. These additional penalties for social actions reflect the penalty for "bothering the user", as well as the time it takes to execute these actions.
- Observation $o \in \Omega$ represents an observation generated by the human. These are tuples of language and gesture: $\langle l, g \rangle$. Language utterance l is represented by a string of any number of words, obtained by transcribing microphone input using webkit's speech recognition API. A gesture g is represented by a vector from the participant's shoulder to their wrist, and all gestures are interpreted as a straight-armed point. This vector is obtained using the Kinect's tracking software.
- $O(o, s, a) = p(o|s, a)$ describes the probability of of seeing an observation o from the human given their state s and the robot's last action a (though we do not make use of this last parameter). We choose an observation function that reflects that the human is an agent attempting to communicate which object they desire to the human, and thus chooses to generate observations that are more likely to result in the robot delivering the correct object. This is gone into more detail in section 4.3.1

4.3.1 Observation Function

According to our double-agent model, the human emits observations as though it were an agent interacting with our robotic agent. Thus, we choose an observation model that depends on the human's belief about the robot's state, β. Specifically, the human will choose an action according to the its estimate that the robot will hand them that object. In order to define this function, we will first have to define a base-level observation function.

Base-Level Observation Function The base level observation function describes the probability of an observation conditioned only on the object: $p(o|\iota)$. For our object delivery domain, we will define two base-level observations, one for language and one for gesture.
Speech Model: Language is interpreted according to a unigram gesture model. An utterance l is broken down into individual words, $w \in l$:

$$p(l|\iota) = \prod_{w \in l} p(w|\iota) = \prod_{w \in l} \frac{\texttt{count}(w, \iota.\texttt{vocab})}{|\iota.\texttt{vocab}|} \tag{1}$$

where $\texttt{count}(w, \iota.\texttt{vocab})$ is the number of times word w appears in ι's vocabulary.

Gesture Model: All gestures are interpreted as a straight armed point. These pointing gestures are selected from a normal distribution centered at the object's location.

Define the angle between the vector defined by the pointing gesture and the vector from the human's arm to the object ι to be θ_ι. The probability of a particular gesture is then

$$p(g|\iota) = \mathcal{N}(\theta_\iota|0, v) \tag{2}$$

where v is a hand-tuned variance.

Posterior Observation Function We will use the base-level observation functions defined above to define a posterior observation that considers the effects of the base level observation function. Specifically, the human chooses an observation proportional to the robot's belief in the desired object if the human had chosen that observation, i.e.,

$$p(o|s) = p(o|\iota, \beta) \triangleq \eta p(\iota|o)_\beta \tag{3}$$

$$= \eta \frac{p(o|\iota)p(\iota)}{\sum_{\iota'} p(o|\iota')p(\iota')} \tag{4}$$

Toy Example

4.3.2 Transition Function

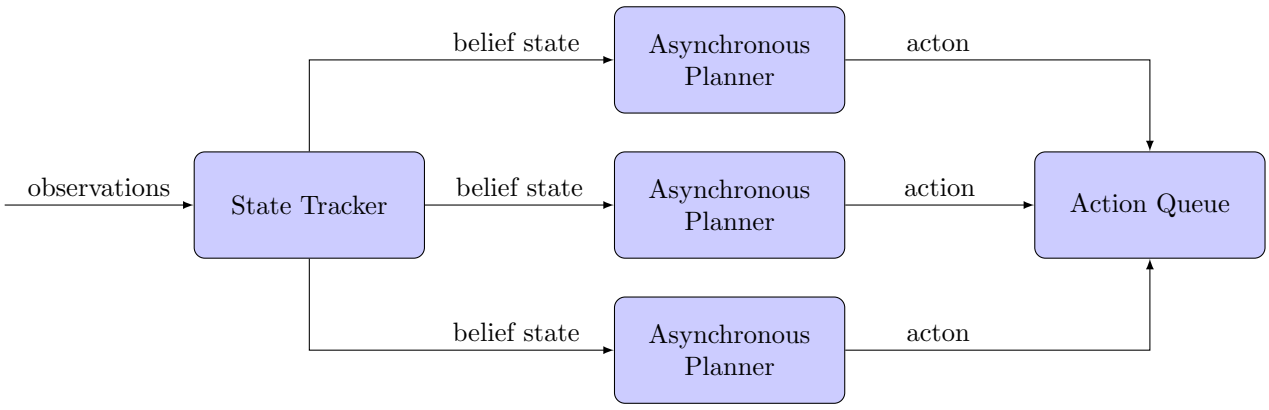
4.4 Policy Generation

4.5 Incremental System Design

Interactive systems that involve speech often make an assumption that the interaction can be split neatly into turns: first one speaker provides a complete utterance, and then the other. This is often not the case, and it is a subtle matter to determine when one turn is completed and the other agent can begin to speak. As a consequence, many systems involving dialogue, including earlier iterations of this project, do not "hear" utterances or gestures while the agent is speaking or acting. Any speech or gestures the human made while the robot was executing its action were dropped and never incorporated into the robot's belief about the state. An additional problem was that while the robot was planning its next action, it would also not incorporate any observations made. This was a significant affect, as it could take up to 10 seconds of planning for the robot to make its next move. As a consequence, most observations would be dropped unless we incorporated a system of turn-taking where we waiting a set amount of time for the human to provide any speech and gesture input.

While turn-taking is an adequate solution for dialogue-only systems, full fledged interaction is more continuous. Even when speaking in a turn-like manner, agents give signals back and forward to signal their understanding and also provide corrections and interruptions.

In order to create more fluid interactions, we employ a multithreaded solution that separates the state estimation, planning, and executions of actions. This allows our robot to simultaneously track the human's belief, plan its next move, and interact with the human.



State Tracker The state tracker incorporates observations from the human at a very high rate, allowing the tracker to always have access to the most up-to-date state estimation. However, our domain description requires the robot to choose an action between each consecutive observation. We address this by feeding the model a `wait` action unless the planning thread has provided a action that has not yet been integrated, in which case we use that action. This also requires minimal dependency between actions and observations, which our model avoids.

Asynchronous Planner A separate thread runs the planning code. Planning can take several seconds (though less is ideal), during which the robot is not only idle, but also deaf and blind. Running the planning code on a separate thread allows the robot to continuing observing and interacting with the robot while the robot plans its next move. In addition, we can use several planning threads at once, with separate policies to control the robot's face, arms, head angle, etc. In our work, we use one thread to plan for the POMDP domain described above, and another to animate the Baxter's face to show an facial expression of increasing confusion with the entropy of the belief state.

Action Queue The planning threads push their actions onto a queue which executes them one by one, allowing the planning threads to resume planning if the actions they produce take some time to execute. An additional advantage is that this allows the robot to be constantly acting, with minimal idle time.

5 Evaluation

5.1 User Studies

6 Future Work