A modular reinforcement learning model for human visuomotor behavior in a driving task

January 10, 2011

Abstract

We present a task scheduling framework for studying human eye movements in a realistic 3D driving simulation. Human drivers are modeled using a reinforcement learning algorithm with "task modules" that make learning tractable and provide a cost metric for behaviors. Eye movement scheduling is simulated with a loss minimization strategy that incorporates expected reward estimates given uncertainty about the state of environment. This work extends a previous model that was applied to a simulation of walking; we extend this approach using a more dynamic state space and adding task modules that reflect the greater complexity in driving. We also discuss future work in applying this model to navigation and fixation data from human drivers.

1 Introduction

Humans formulate and execute complex visuomotor action sequences while performing real-world tasks like driving or playing sports. Previous work has explored the role that visually "salient" [5] features play in making saccades, but this research has focused largely on 2D images or videos where human subjects are observing the scene and not actively participating in a visuo-motor task. In contrast, when performing tasks in natural environments, humans interact with the world, and high-level cognitive goals and reward [3, 6] play an important role in the execution of eye movements. However, the mechanisms underlying these task-driven saccades are not well understood.

This paper presents a high-level, task-based scheduling framework for studying human eye movements in a realistic, 3D driving simulation. Our primary aim is to present an abstract framework for interpreting human eye movement behavior that explicitly represents task demands, reward and perceptual uncertainty. This approach allows modeling of visual behavior over long time scales that has not been typically addressed in vision science. Our model is quite abstract in that no image processing is used and major simplifications are made to ease the process of modeling driving behavior. The model is still in development and we focus here on providing a technical report of our methodology and a review of the current state of ongoing research.

We model human drivers computationally using a reinforcement learning algorithm that breaks the complex state space of driving into several "task modules" that make learning computationally tractable [7]. Modules also provide a cost metric that allows direct comparison of the relative values of different behaviors. In our model, eye movement scheduling attempts to minimize the expected loss of reward given the current knowledge of the state of the world and the uncertainty in the state estimate. From a high level, eye movements are directed toward targets in order to reduce uncertainty about potentially high-reward portions of the state space.

2 Background

Prior research suggests that although human vision has massively parallel inputs from the retina, due to attentional and memory limitations many visuo-motor computations are serial [3]. Studies have found that humans often employ active vision strategies of gathering specific and discrete pieces of visual information as a task develops [4, 2]. These data suggest that one approach to model goal directed human vision is to use serial "visuo-motor task modules," sometimes referred to as visual routines [1, 11]. These modules perform very specific computations in isolation (e.g., finding a road landmark to control steering), but when coordinated over time with other modules, complex behaviors can be achieved.

With this type of approach, a scheduling problem arises: What visuo-motor computations should be carried out and when should they be executed? Our work presents one solution to this scheduling problem by using reward values and uncertainty to solve the arbitration of visual computations. This work extends a previously developed reinforcement learning model [9] that has been successfully applied to a simulated three-task walking world with static obstacles and goals.

Sprague and Ballard simulated a humanoid walking down a sidewalk with obstacles and "litter" to be picked up. Their algorithm has distinct perceptual and motor components. Visual computations were broken down into components for avoidance of obstacles, "picking up" items and sidewalk following. Each of these modules has a dedicated visual computation that finds the distance and angle to the sidewalk, obstacles and litter. The motor system uses this state information to navigate (turn left, turn right or go straight) using a control policy learned via reinforcement learning. Only one visual module at a time can run to get a new update of state information. Idle modules are allowed to update their representations via a Kalman filter, introducing uncertainty into their state estimates. The perceptual arbitration system selects a module to be updated with new sensory information. Crucially, the perceptual arbitration algorithm uses reward estimates from the motor component and estimates of state uncertainty in the perceptual system to choose which module to update.

The present research applies a similar methodology to a simulated driving task. In comparison to walking, the driving task requires a more complex and dynamic state space and has more task modules to address the greater variety of available tasks while driving. After briefly introducing reinforcement learning and describing the driving simulation, we present some preliminary results and then conclude with a description of future work in applying this model to navigation and fixation data from human drivers in a realistic 3D car simulation.

3 Reinforcement Learning

Reinforcement learning (RL) [10] is a goal-focused learning framework that directly models the interaction between learner and environment. RL finds a mapping between a current environmental state and an appropriate action to execute in that state. In our application, we use RL to find a control policy that maps environmental states to actions to control steering and velocity of a simulated car. Our specific implementation of the state and actions spaces is presented in section 4.

Here we present a brief primer on the RL framework, focusing on the Q-learning [12] variant of the algorithm. A learning agent (LA) maintains a vector s_t of discrete variables describing the state of the world over a series of discrete time steps $t = 1 \ldots T$. At each time step, the LA chooses a discrete action a_t that will maximize the available reward. Positive or negative reinforcement r_t is given to the LA whenever s_t is a state that achieves some goal or subgoal, specified by the modeler as part of the construction of the world. The LA receives supervision only in the form of these explicit reward values,

which are often nonzero only for a small fraction of world states.

During training, the LA constructs an exhaustive Q table $Q(s_t, a_t)$ of the expected rewards that are attainable by taking each action from each state in the world. If the LA takes an action a_t when the world is in state s_t, it observes the resulting state s_{t+1} and its associated reward r_{t+1} on the following time step. Using a learning rule, the LA can then update $Q(s_t, a_t)$ so that over time this Q value becomes closer to the "expected future reward" for (s_t, a_t). The LA adjusts the Q values by following the gradient of the error in Q:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \Delta Q(s_t, a_t)$$

where $\alpha \in [0, 1]$ is a learning rate parameter (set to 0.2 for our simulations) and ΔQ is the direction of the greatest observed change in Q at (s_t, a_t).

The optimal Q values reflect both the immediate reward in a state, and all future rewards attainable from that state, discounted exponentially by the number of time steps required to reach those future states. Thus ΔQ takes the form

$$\Delta Q(s_t, a_t) = r_{t+1} + \gamma \hat{Q}(s_{t+1}) - Q(s_t, a_t)$$

where r_{t+1} is the reward available in state s_{t+1} (which follows state s_t after taking action a_t) and γ is a parameter called the discount factor. Values of γ near 0 cause the LA to rely more on immediate rewards for the Q values, while values near 1 blur the distinction between immediate and future reward, allowing the agent to postpone immediate rewards for potentially larger future rewards. For our simulations, we set $\gamma = 0.999$.

A simple yet powerful learning rule for $\hat{Q}(s_{t+1})$ is simply the Q value associated with the subsequent action selected by the agent:

$$\hat{Q}(s_{t+1}) = Q(s_{t+1}, a_{t+1})$$

This rule, called SARSA learning, ensures that, along any given sequence of state/action pairs that are actually chosen by the agent, the expected rewards obey the discounting enforced by the γ parameter.

Our simulation uses the Q values to choose an action using a softmax rule, where the probability of choosing action a_t when the world is in state s_t is given by

$$p(a_t|s_t) = \frac{\exp\left(Q(s_t, a_t)\right)}{\sum_a exp\left(Q(s_t, a)\right)}.$$

3.1 GM-SARSA

Traditional RL operates within a single, joint state space that must be capable of representing all task-relevant aspects of the world simultaneously. Because the LA must visit each state/action pair multiple times during learning to formulate an accurate estimate of the Q values, a large state space leads to slower convergence during learning. In complex environments, RL is much more efficient if a learner is allowed to focus just on the state variables that are relevant for a particular task. Instead of running the driving simulation in a joint state space that represents all possible variables of interest simultaneously, we used a technique called GM-SARSA [8] to split the world into small task modules.

In GM-SARSA, each task module $i = 1 \ldots N$ has a separate state space and Q table, $Q^i(s_t^i, a_t)$, but the tasks share a common action space. When the LA needs to select action a_t, it uses the state estimates $s_t^1, \ldots, s_t^N$ for each task to retrieve the corresponding vectors of Q values $Q^1(s_t^1, \cdot), \ldots, Q^N(s_t^N, \cdot)$. These vectors are summed, and the result

$$Q^*(a_t) = \sum_{i=1}^{N} Q^i(s_t^i, a_t)$$

is used in the decision rule to select the best action.

The SARSA learning rule maintains the correctness of task learning with multiple modules. Because the action a_t is shared among all modules, the Q tables can be updated correctly even though a_t might not have corresponded to the

highest-reward action for any of the individual task modules.

4 Modular RL for Driving

Our driving model consists of $C \approx 20$ cars that drive in the lanes of a simulated world including a four-lane road (two lanes in each direction), cars, and pedestrians. Two of the cars in the world have special roles: car 1 is controlled by the learning agent in the simulation, and car 2 is called the "pace car" and is described in more detail below. Cars $3 \ldots C$ serve mostly as obstacles for the learning agent. Figure 1 shows a screenshot of the cars in the simulated world, and figure 4 shows a screenshot of state space in the simulated world, after we have projected it into the realistic, 3D driving environment that we use for human subjects.

All of the cars move in the same direction and are constrained to drive along one of two tracks that represent the two available lanes on the road. Each car thus maintains three scalar variables that describe its state in the world: δ_c represents the distance (in meters) traveled along the track by car c, σ_c represents the speed (in meters per second) of the car on its track, and $\lambda_c \in \{0, 1\}$ represents the lane that car c currently occupies. In the 3D simulation for humans, described in more detail below, these scalars are mapped to the 3D positions of the lanes in a virtual world that also includes buildings, signs, and other realistic effects, but each car in the RL portion of the simulation is completely represented by these three scalars.

All agents other than the learner move at a fixed speed along one track, but these states change randomly on average every 1000 time steps to prevent the LA from overlearning a static world. When choosing new values, cars $2 \ldots C$ draw a new speed uniformly from $[0, \Sigma]$ (where Σ is the maximum speed allowed for any car, set to 13 m/s in our simulations) and a new track uniformly from $\{0, 1\}$.

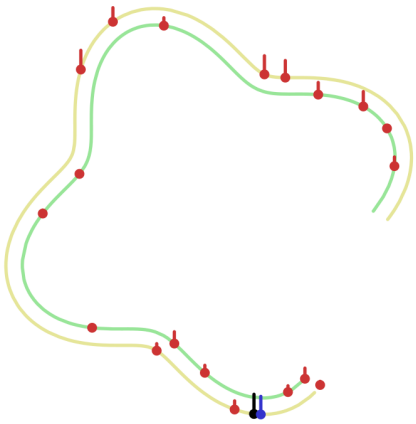

Figure 1: The simulated RL driving world consists of the LA (black dot), a pace car that the LA is rewarded for following (blue dot), and several other agents that the LA is punished for hitting (red dots). Each car has a "flag" whose length indicates the car's speed. Lanes for driving are shown as curvy colored lines, even though to the RL agents the lanes are one-dimensional.

4.1 Task Modules

The RL model uses several modules coordinated over time to drive. While there are many possible sub-tasks to include in our framework we limit ourselves to a basic set that could be applied to data from human drivers. These modules are dedicated to tasks for avoidance, following another car, and simply driving forward. Figure 2 shows a graphical representation of the state space used by each module discussed below. Additionally, Figure 3 (page 8) shows a high level overview of the scheduling model and how modules are coordinated.

4.1.1 Forward Progress

The LA is encouraged to move around the track by a dedicated task module that provides a small positive reward R_Σ whenever the LA is moving at speed greater than $\frac{\Sigma}{2}$. Without this task module, the LA tends to stop moving, which no humans do in the 3D driving simulator. The state

space for this task is simply the speed of the LA, divided into N_σ uniformly spaced bins.

4.1.2 Car Following

The LA receives a positive reward R_f for following the pace car at a fixed distance of 10 m, with a relative speed of 0 m/s (i.e., whenever the LA is following behind the pace car and both cars are going the same speed). The state space for this module consists of three dimensions: the lane indicator, the relative distance, and the relative speed. The lane indicator is an ordered pair from $\{0,1\} \times \{0,1\}$ that represents the lanes for the LA and the pace car. The relative distance is given by $\max(\min(\delta_2 - \delta_1, D), -D)$, where D is a constant (set to 200 m in our simulations) that represents the maximum distance the LA can discriminate. This dimension is quantized into N_δ uniformly spaced bins. Similarly, the relative speed is given by $\max(\min(\sigma_2 - \sigma_1, \Sigma), -\Sigma)$ and is quantized into N_σ uniformly spaced bins.

4.1.3 Car Avoidance

The LA receives a negative reward R_c for colliding with any of the other cars in the world. A world state is considered a collision whenever

$$|\max(\min(\delta_* - \delta_1, D), -D)| < \frac{2D}{N_\delta}$$

and the relative speed between the LA and the obstacle is less than 0. This task uses the same state space as the following task described above. The driving simulation includes one task module that tracks the closest obstacle (including the pace car) to the LA at every time step, but could easily include more such modules representing the states of the next-closest obstacles.

4.2 Action Space

While each module tracks different information in the world, they share the same set of actions. Given the current state of the world, the LA can read out the Q estimates for each module and evaluate the optimal action via GM-SARSA. The action space for each module contains a steering component and a velocity component. The actions are discretized such that steering control has three options: staying in the same lane, changing to the right lane, or changing to the left lane. Similarly, velocity control features three options: speed up, stay at the same speed, or slow down.

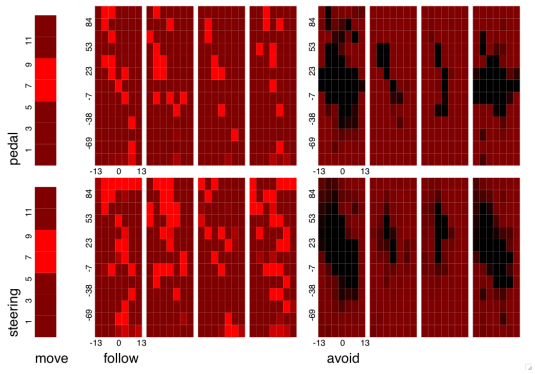

Figure 2: The state space of each of the three simulated tasks (arranged in columns: move, follow, and avoid) and two action variables (arranged in rows: pedal and steering). Within the left column, the "forward progress" task uses only the speed of the LA. In the middle, the "follow" task uses the lanes of the LA and the pace car (major horizontal axis), their relative speeds (minor horizontal axis), and their relative distances (vertical axis). Within the right column, the "avoid" task uses the same state space as the "follow" task, but represents the distance and speed to the closest obstacle in the world. Shades of red in this diagram depict the Q values for corresponding states: dark shades represent negative values, while bright shades represent positive values.

4.3 Training and Evaluation

Training takes place in episodes that start with a randomly configured world and end whenever any task module of the LA has been in the same state for 10 time steps. When an episode ends,

the LA undergoes 100 time steps of evaluation to track the progress of training, and then the world is reconfigured randomly and a new training episode begins.

To reset the world, all cars $2 \ldots C$ are placed by randomly selecting a lane, position, and speed from uniform distributions across the full ranges of these variables. The LA is placed behind the pace car, but the distance from the LA to the pace car increases in variance with the number of training episodes. This gives the LA early exposure to the high-reward state that follows the pace car around the track, but makes the following task increasingly distant in state space as training progresses.

During evaluation, the world is reset to a "test state" and 100 time steps of simulation are performed. The LA uses its Q tables to calculate the optimal action at each time step, but no learning takes place. The number of time steps in which the learner is following the pace car, along with the number of time steps in which the LA intercepts another car (i.e., a collision) are recorded.

5 Eye Movements

5.1 Perceptual Arbitration

In the learning framework we have presented so far, the LA always has access to accurate state information. This is not the case in a real driving task, where a human driver with limited visual resources must fixate specific targets over time to resolve their true locations or speeds. Therefore, we follow the approach developed for a more static walking task [9] and incorporate the notion of state uncertainty into our model.

Instead of making a decision based on perfect state knowledge, the LA maintains an estimate of $\tilde{s}_t^i$ for each task module i in the driving simulation. This state estimate consists of a probability distribution over the entire state space; the most likely state of the world corresponds to the mode of this distribution, but the world might have changed since the LA last took an accurate state measurement (e.g., by foveating some object like the pace car).

When choosing an action, the LA multiplies its state estimate distribution with the learned Q tables, yielding an expected reward metric. For a given task module b, the loss ℓ^b incurred for not updating a module's state estimate is the difference in expected value between the reward that the LA might receive if it had perfect state information and the estimated value of the reward $\tilde{Q}^b$ given the action a^* that would be selected by the current (imperfect) state information:

$$\ell^b = E\left[max_a\left(Q^b(s^b,a) + \sum_{i\neq b}\tilde{Q}^i(s^i,a)\right)\right] - \sum_i \tilde{Q}^i(\tilde{s}^i,a^*)$$

This loss function can be used to guide the LA's perceptual resources during a simulation. Figure 3 (page 8) shows a diagram of the information flow in the computational model.

6 Future Work

The computational modeling work described in this paper forms part of a larger attempt to quantify and analyze human visual behavior in a realistic driving task. The model is in development and we are currently working to improve learning and add additional behaviors for dealing with pedestrians and oncoming cars. Additionally, because the model provides quantitative costs for various actions that the LA can take in the world, a major focus of our future work is to use the model to provide a plausible mechanism for explaining eye movements of human subjects navigating in a world involving multiple distinct tasks.

Our lab has a virtual reality driving simulator, consisting of driving platform with pedals and a steering wheel, a head tracking system, and a head mounted display (HMD). An eye tracker is mounted on the HMD. Preliminary data has been collected from human subjects driving in

Figure 4: The state space of the RL simulation can be projected easily into a 3D virtual reality driving simulation in the lab. Human subjects see this sort of view of the driving environment as they drive around in the virtual world.

a realistic urban environment with a pace car, other cars and pedestrians present; see Figure 4 for an example screenshot from the environment.

Preliminary analysis of human fixation data suggests that distributions of fixations are inconsistent with a simple scheduling models (e.g. round robin), suggesting a scheduler like the one presented here may have more utility. While the current application of our methodology to driving is still in development, we believe that this general framework is a powerful and unique approach to understanding human vision and may also have broader application in the construction of computer vision systems.

References

[1] D.H. Ballard, M.M. Hayhoe, P.K. Pook, and R.P.N. Rao. Deictic codes for the embodiment of cognition. *Behavioral and Brain Sciences*, 20(04):723–742, 1997.

[2] J.A. Droll, M.M. Hayhoe, J. Triesch, and B.T. Sullivan. Task demands control acquisition and storage of visual information. *Journal of Experimental Psychology*, 31(6):1416–1438, 2005.

[3] M.M. Hayhoe and D.H. Ballard. Eye Movements in Natural Behavior. *Trends in Cognitive Sciences*, 9(4):188–193, 2005.

[4] M.M. Hayhoe, D.G. Bensinger, and D.H. Ballard. Task constraints in visual working memory. *Vision Research*, 38(1):125–137, 1998.

[5] L. Itti and C. Koch. Computational Modelling of Visual Attention. *Nature Reviews Neuroscience*, 2(3):194–203, 2001.

[6] V. Navalpakkam, C. Koch, A. Rangel, and P. Perona. Optimal reward harvesting in complex perceptual environments. *Proceedings of the Nationalal Academy of Sciences*, in press.

[7] C.A. Rothkopf and D.H. Ballard. Credit assignment in multiple goal embodied visuomotor behavior. *Frontiers in Psychology, Special Topic: Embodied and grounded cognition*, 2010.

[8] N. Sprague and D.H. Ballard. Multiple-Goal Reinforcement Learning with Modular SARSA(0). In *International Joint Conference on Artificial Intelligence*, volume 18, pages 1445–1447. Citeseer, 2003.

[9] N. Sprague, D.H. Ballard, and A. Robinson. Modeling embodied visual behaviors. *ACM Transactions on Applied Perception*, 4(2), 2007.

[10] R.S. Sutton and A.G. Barto. *Reinforcement learning: An introduction*. The MIT press, 1998.

[11] S. Ullman. Visual Routines. *Cognition*, 18(1-3):97–159, 1984.

[12] C.J.C.H. Watkins and P. Dayan. Q-learning. *Machine learning*, 8(3):279–292, 1992.

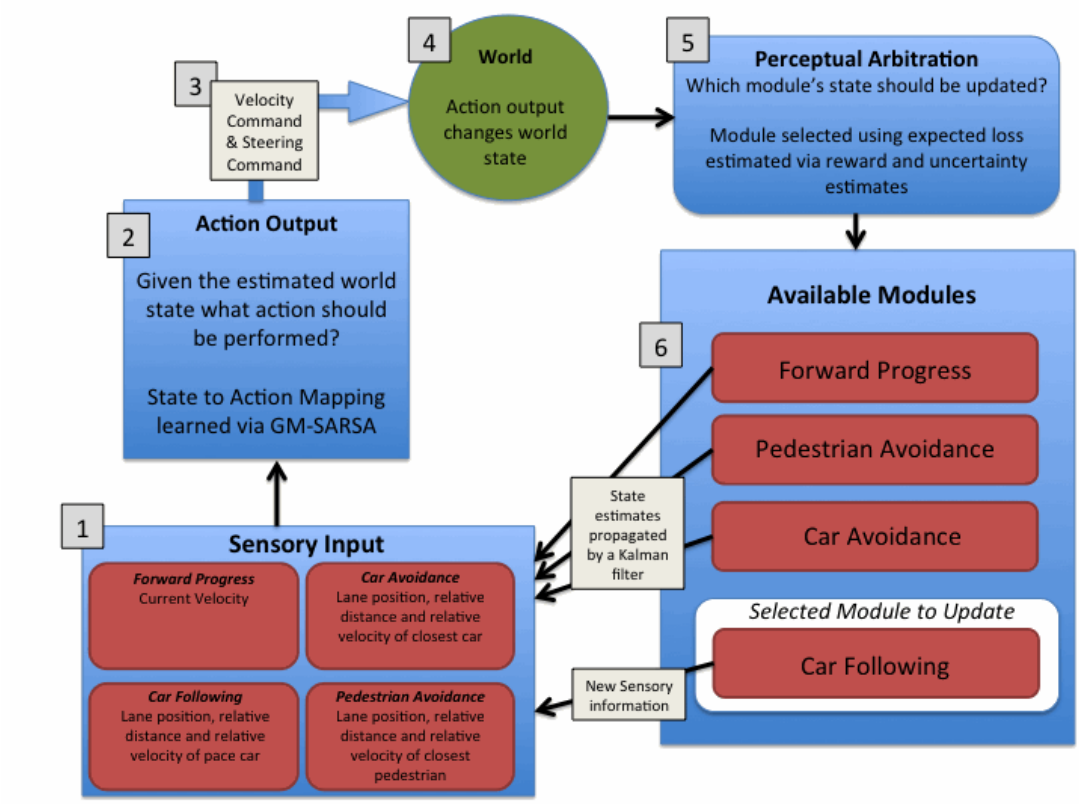

Figure 3: Flow diagram of the task-module scheduling architecture. Following the numeric labels, (1) the system initializes with a set of sensory readings about the world. Each task module has a representation of their state space that is (2) mapped to a learned action policy via the GM-SARSA algorithm. This mapping allows the driving agent to (3) output a steering and velocity command to drive the car. These actions (4) take have some effect in the world that changes the world state. (5) Using information on potential rewards and state uncertainty, the perceptual arbitration algorithm chooses the module most in need of update to its world state estimate. (6) In this example the Car Following module is chosen To be updated and is able to gain access to new sensory information. The other modules cannot update and are forced to propagate estimates of their world state using a Kalman filter. This perception and action loops repeats itself each time step as the driving agent traverses through the environment.