Eye Movements & Cognitive Models

Dario Salvucci

Drexel University
Eye Movements & Cognition

- Eye movements help us understand cognition
  - What is this driver thinking?
  - Could you guess the driver’s intentions without them?
Eye Movements & Cognition

- Cognition helps us understand eye movements
  - Why does the driver look at the lead car?
  - Why doesn’t the driver look at oncoming traffic?
Eye Movements & Cognition

- Models help us understand both
  - How does is gaze information being acquired and used?
  - What information is being captured without gaze?
Eye Movements & Cognition

- Eye movements help us understand cognition
  - Without eye movements, we would see…
    - a long delay
    - a keyed response
  - What strategy is being used to solve the equation?
Eye Movements & Cognition

- Cognition helps us understand eye movements
  - Sometimes, eye movements don’t behave as expected
  - How do we explain “misplaced” or missing fixations?
Eye Movements & Cognition

- Models help us understand both
  - They give us a way to tie the eyes and brain together

![Diagram](Image)
Eye Movements & Cognition

- Cognition, eye movements use the same processes and mechanisms in all these domains
- So we should explain them with a unified set of processes and mechanisms
Cognitive Architectures

- Cognitive architecture = psychological theory + computational framework
  - like a human-modeling programming language
  - built-in “functions”
    - e.g., memory store and recall, goal and subgoal setting, perceptual-motor behavior
  - built-in limitations
    - e.g., forgetting, errors, perceptual-motor parameters

→ Models are constrained within larger theory
→ Models are psychologically plausible
Cognitive Architectures

Domain “Software” knowledge for individual domains

Brain “Hardware” core mechanisms that all people have
ACT-R

Goal Module (not identified)

Goal Buffer (ACC)

Procedural Module (Basal Ganglia)

Declarative Module (Temporal/Hippocampus)

Retrieval Buffer (VLPFC)

Visual Buffer (Parietal)

Manual Buffer (Motor/Cerebellum)

Declarative Module: factual knowledge as “chunks” of information

Goal Buffer: directs procedural module to goal-relevant actions

Procedural Module: procedural skill as goal-directed production rules

External World

Visual Module (Occipital/Parietal)

Manual Module (Motor/Cerebellum)

Goal Module (not identified)

Goal Buffer (ACC)

Procedural Module (Basal Ganglia)

Declarative Module (Temporal/Hippocampus)

Retrieval Buffer (VLPFC)

Visual Buffer (Parietal)

Manual Buffer (Motor/Cerebellum)

Declarative Module: factual knowledge as “chunks” of information

Goal Buffer: directs procedural module to goal-relevant actions

Procedural Module: procedural skill as goal-directed production rules

External World

Visual Module (Occipital/Parietal)

Manual Module (Motor/Cerebellum)
ACT-R Vision

- **Spotlight theory of attention**

  - limited fovea of high resolution
  - large periphery of degraded resolution

what the world looks like

how we see the world
ACT-R Vision

- **“Where”** process is basically constant time
- **“What”** process depends on what’s being encoded
  - e.g., *the* vs. *antidisestablishmentarianism*
  - e.g., *hän* vs. *peruspalveluliikelaitoskuntayhtymä*
  - (but general visual objects are a challenge: △? ⇔?)

Find a location in my visual field that satisfies a set of constraints

Given that I have the location, move my attention to that location

Use the encoded object
EMMA dictates how visual attention ≠ fixation

- ACT-R model generates a shift of attention
- soon after (~200 ms) — maybe — the eyes move to the target of attention [based on E-Z Reader]
  - “labile” stage that can be cancelled if a new shift occurs
  - “non-labile” stage that cannot be cancelled
- but…
  - movement can miss target, requiring a re-fixation
  - attention can shift quickly again, skipping the first target

\[
\frac{5 \times 28}{15} = \frac{15}{7}
\]
Driving as a Single Task

- Two-level steering with near & far points

- ACT-R model procedural steps

![Diagram showing two-level steering with near and far points and ACT-R model procedural steps with visual and procedural updates.]
Driving as a Single Task
Driving as a Single Task

- Curve negotiation

![Graphs showing steering angle and lane position for human and model](image)

**Steering Angle**

**Lane Position**

Human

Model
Driving as a Single Task

- Lane changing

![Graphs showing steering angle and lane position for humans and models during lane changing.](image)
Driving as a Single Task

- Gaze time on regions of interest

![Graph showing proportion of gaze time on different regions in current and other lanes.](chart.png)
Driving as a Single Task

- Human driver...
Driving as a Single Task

- Model driver...
A Word about Abstraction

- At what “level of abstraction” should we model?
  - individual fixations and saccades
  - individual gazes
  - aggregate gaze time on regions
  - or higher levels? or lower levels?

- Many different levels may be valid, and useful
Modeling & Programming

- Lots of work in the 1980s-90s on modeling & programming within the ACT-R architecture
- Practically all of it was done in the context of Intelligent Tutoring Systems for programming
- Some of this work includes...
  - Model + tutor for programming recursion (Pirolli et al.)
    - “students can learn very different rules for recursive programming from the same example programs”
  - ACT Programming Tutor (Corbett et al.)
    - knowledge tracing: keeping a “skill-o-meter”
  - [Also: Eye movements in (math) tutoring systems]
Intelligent Tutoring Systems

- **ITS =** system that tracks student cognition and teaches accordingly

- **Central idea: Model tracing**
  - that is, relating observed actions with unobservable cognitive states
  - in essence, “think” along with the student and keep track of cognitive state
    - simulate all possible “thought” sequences
    - find which model sequence matches human behavior
    - make the best matching sequence the current estimated cognitive state
    - “cognitive state” = state of the cognitive model
Model Tracing

Cognitive Model

Observed Actions

conflict resolution

rule firing

Model Tracing

Cognitive Model

Observed Actions

“buggy” rule
ACT Programming Tutor (Corbett et al.)
ACT Programming Tutor (Corbett et al.)

Problem Statement

Define a lisp function named last-item that takes a list as an argument and returns the last element of the list. For example,

(last-item '(a b c d e f)) returns f
(last-item '(w x y z)) returns z

Lisp Exercise 1.87 Last-Item

(defun LAST-ITEM (LIS)
  (car <EXPR1>)
  <PROCESS1>)

<EXPR0>
ACT Programming Tutor (Corbett et al.)

- Extract an embedded list
- Extract info from an embedded list
- Extract info from a list
- Deleting an extra node from the parameter list
- Coding a variable
- Declaring a function parameter
- Coding a function name
- Coding DEFUN
- Remove N Items
- Skip over Items
- Work From the Back of the List
- Extract the Last Item
- Extract the Nth Item
- Coding LIST - embedded lists involved
Concert tickets cost 45 dollars a piece. A friend offers to stand in line to buy a number of tickets, if you will pay him a fee of 12 dollars to do so.

Under this arrangement, how much would 5 tickets cost?

What would be the total cost of 8 tickets?

For the formula, define a variable for the number of tickets, and use this variable to write a rule for the cost.
**Eye Movements & Tutoring (Gluck)**

A hot air balloon is at an altitude of 75 feet. With time, the passengers get bored and decide to land the balloon. They descend at 6 feet per minute.

**Question:**

1. What altitude is the balloon after 3 minutes have passed?
2. How high are they 7 minutes after they start to descend?

**Formula**

For the formula, define a variable for the time since they started to descend, and use this variable to write a rule for the altitude of the balloon.

<table>
<thead>
<tr>
<th>Unit</th>
<th>time (minutes)</th>
<th>altitude (feet)</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>75 - 6 = 3</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Help | Done
Eye Movements & Tutoring (Gluck)

- Eye movements helped to understand...
  - Failure to read bug messages
  - Disambiguation of an error
  - Time off-task
  - and more

Key: Looking for “instructional opportunities” that would not be present without eye movements

- Human tutors can have a huge impact (e.g., raise a student 2 standard deviations from mean!)
- Computer tutors can manage about 1 standard deviation
  - maybe measures like eye movements can help close the gap
Intelligent Tutoring

- So, there were models of programming built into these intelligent tutors
- And there have been tutors that (sort of) use EMs
- But interestingly…
  - there are many tutors today (for 300,000+ students)
  - but no (ACT-R) programming models since 1990s!
  - tutors all based on 1990s cognitive architecture, with 1 rule \( \approx 1\text{-}5\) seconds of action
    - modeling behavior at a more strategic level
  - modern architectures: 1 rule \( \approx 50\text{-}250\) ms of action
    - good at predicting eye movements, but harder to build tutors at the strategic level
Modeling & Programming

- What lessons might we draw from ACT-R’s history of modeling, tutoring, and eye movements?
  - Modeling the cognitive processes in programming is difficult, but can be done.
    - the cognitive architecture can help guide representations and skill sets to those that are psychologically plausible
  - There is sometimes a large leap from eye-movement patterns to cognitive strategies.
    - Model Tracing is very relevant and may help here
    - this is where the levels of abstraction come in… at what level are you most interested in learning about?

- So how do we map eye movements to strategies in computer programming?
### EMIP 2014 coding scheme

<table>
<thead>
<tr>
<th>Patterns (observable)</th>
<th>Strategies (unobservable)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flicking</td>
<td>AttentionToDetail</td>
</tr>
<tr>
<td>JumpControl</td>
<td>DataFlow</td>
</tr>
<tr>
<td>JustPassingThrough</td>
<td>Debugging</td>
</tr>
<tr>
<td>LinearHorizontal</td>
<td>Deductive</td>
</tr>
<tr>
<td>LinearVertical</td>
<td>DesignAtOnce</td>
</tr>
<tr>
<td>RetraceDeclaration</td>
<td>FlowCycle</td>
</tr>
<tr>
<td>RetraceReference</td>
<td>Inductive</td>
</tr>
<tr>
<td>Scan, Signatures</td>
<td>Interprocedural-ControlFlow</td>
</tr>
<tr>
<td>Thrashing</td>
<td>Intraprocedural-ControlFlow</td>
</tr>
<tr>
<td>Word(Pattern)-Matching</td>
<td>StrayGlance</td>
</tr>
<tr>
<td></td>
<td>TestHypothesis</td>
</tr>
<tr>
<td></td>
<td>Touchstone</td>
</tr>
<tr>
<td></td>
<td>Trial&amp;Error</td>
</tr>
<tr>
<td></td>
<td>Wandering</td>
</tr>
</tbody>
</table>
Modeling & Programming

- Possible patterns/low-level strategies:
  - read everything (syntax checking?)
  - read words (not punctuation)
  - read method names, speak them out

- Models have...
  - ACT-R rules that specify the strategy
  - ACT-R architecture that drive the underlying processes (e.g., visual attention shifts leading to eye movements)

```java
public class Rectangle {
    private int x1, y1, x2, y2;
    public Rectangle ( int x1, int y1, int x2, int y2 ) {
        this.x1 = x1;
        this.y1 = y1;
        this.x2 = x2;
        this.y2 = y2;
    }
    public int width () {
        return this.x2 - this.x1;
    }
    public int height () {
        return this.y2 - this.y1;
    }
    public double area () {
        return this.width ( ) * this.height ( ) ;
    }
    public static void main ( String [ ] args ) {
        Rectangle rect1 = new Rectangle ( 0 , 0 , 10 , 10 );
        System.out.println ( rect1.area ( ) );
        Rectangle rect2 = new Rectangle ( 5 , 5 , 10 , 10 );
        System.out.println ( rect2.area ( ) );
    }
}
```
Conclusions

- Eye movements help us understand cognition
  - BUT they are not the only useful source of data, and are best viewed as complementary with other data

- Cognition helps us understand eye movements
  - BUT the hidden connection between the eyes and the mind will always make this a non-trivial process

- Models help us understand both
  - BUT there are many types of models — all of them are “wrong” :) but many of them are useful!
  - plus, models can be used for engineering development of model-based systems (like tutoring systems)
ACT-R Modeling

- If you’d like to see/run a working model, please download and extract...

  http://cog.cs.drexel.edu/emip15.zip

- Launch “ACT-R-prog.jar” then open “Program1.actr” and click “Run”:
ACT-R Modeling

- The canonical implementation is in LISP:
  - [http://act-r.psy.cmu.edu](http://act-r.psy.cmu.edu)
  - like many (most?) systems developed since the early years of AI, Cognitive Science

- LISP is a cool language, but it’s often inconvenient
  - environments are uneven across platforms
  - task/interface development is more difficult

- We will try an ACT-R system implemented in Java
  - actually, you don’t need to know Java at all to use it (ACT-R has its own language)
  - though you do need Java to program new tasks
  - your “ACT-R-prog.jar” has a programming task built-in
ACT-R Modeling

Domain “Software”
knowledge for individual domains

Brain “Hardware”
core mechanisms that all people have

Cognitive Architecture

model of doing math problems
model of driving
model of buying groceries
model of washing dishes
model of mowing a lawn
Model Overview

(set-task "prog.Programming")

(sgp
    :emma t
    :visual-num-finsts 300
    :visual-finst-span 300
    :visual-movement-tolerance 1
    :v t
)

(add-dm
    (goal isa read)
)

(goal-focus goal)

(p read*find-line ...
)
public class Programming extends actr.task.Task {
    private String[] TEXT = {
        "public class Rectangle {",
        "private int x1, y1, x2, y2 ;",
        ...
    };

    public Programming() { ... }

    @Override
    public void start() {
        // draw TEXT on the screen using labels
        processDisplay(); // to register items with visual system
    }
}
Model Overview

Define a production “read*find-line”

IF the goal...
  is of type “read”
  and the line is currently empty
  and the visual-location status...
  is free
  with an empty buffer
  and the visual status
  is free
  with an empty buffer

THEN
  start a new visual-location process
  for some visual location
  at the lowest ‘y’ coordinate
  that has not yet been attended

(p read*find-line
 =goal>
   isa read
   line nil
 ?visual-location>
   state free
   - buffer requested
 ?visual>
   state free
   buffer empty
==> +visual-location>
   isa visual-location
   screen-y lowest
   :attended nil )
Model Overview

Define a production “read*note-line”

IF the goal...
  is of type “read”
  and the line is currently empty
  and the visual-location contains a chunk...
  of type visual-location
  at a particular ‘y’ coordinate (=y)

THEN
  change the goal
  to note the found ‘y’ coordinate

(p read*note-line
  =goal>
  isa read
  line nil
  =visual-location>
  isa visual-location
  screen-y =y
==>
  =goal>
  line =y
)
Model Overview

(p read*find-token
  =goal>
    isa read
  line =y
?)

state free
- buffer requested

?visual>
  state free
  buffer empty

==>
  =goal>
  line =y

+visual-location>
  isa visual-location
  screen-y =y
  screen-x lowest
  :attended nil
)

(p read*encode-token
  =goal>
    isa read
    line =y
  ?visual>
    isa visual-location
  state free
  buffer empty

==>
+visual>
  isa move-attention
  screen-pos =visual-location
)

(p read*continue-line
  =goal>
    isa read
    line =y
  ?visual>
    value =text

==>
!output! (=text)
)

(p read*done-line
  =goal>
    isa read
    line =y
  ?visual-location>
    state error

==>
  =goal>
    line nil
  -visual-location>
)
Simulation Run

> (run)

0.000  vision             unrequested  [vision~63]
0.000  procedural         start
0.049  procedural         ** READ*NOTE-LINE  **
0.098  procedural         ** READ*FIND-TOKEN  **
0.098  vision             find-location  [vision~66]
0.147  procedural         ** READ*ENCODE-TOKEN  **
0.147  vision             move-attention
0.168  vision             encoding-complete  [word~69]
0.217  procedural         ** READ*CONTINUE-LINE  **

"public"

0.266  procedural         ** READ*FIND-TOKEN  **
0.266  vision             find-location  [vision~72]
0.282  eye                 preparation-complete  [word~69]
0.315  procedural         ** READ*ENCODE-TOKEN  **
0.315  vision             move-attention
0.356  eye                 execution-complete  [word~69]
0.370  vision             encoding-complete  [word~75]
0.419  procedural         ** READ*CONTINUE-LINE  **

"class"