Overview

Lecture 5
⇒ Focus on motivation, objectives, and design in the JAMES project.

Lecture 6
1. Recall: motivation, objectives, and design of JAMES
2. High-level decision making and planning
3. Artificial Intelligence planning in JAMES
4. Final thoughts: design, HRI, and JAMES
⇒ Technical details about how and why the robot does what it does.

PART I
RECALL: MOTIVATION, OBJECTIVES, AND DESIGN OF JAMES

Task-based social interaction in JAMES

• Robot will play the role of a bartender.
• Must respond to customers’ requests in a dynamic setting with multiple users and short interactions.
• Interactions incorporate both task-based aspects (e.g., ordering and serving drinks) and social aspects (e.g., managing multiple interactions).
• Interactions in German or English.

Image: fortiss GmbH
Socially-appropriate interaction

Two people, A and B, each individually approach a bartender.

Bartender (to A): How can I help you?
Person A: A pint of cider, please.

Person C approaches the bartender and attracts his attention.

Bartender (nods at A, then to C): Just a moment please.
Bartender (Serves A):
Bartender (to B):
Person B: A glass of red wine.
Bartender (to C):
Person C: Thanks for waiting.
Bartender (Serves C):
Bartender (nods at C):
Person C: I’d like a pint of bitter.

Factors influencing the design of JAMES

• Core activities of the robot
  – YES: asking users for drinks, clarifying requests, serving drinks, ...
  – NO: pouring drinks, handling money, small talk, ...

• Results from human experiments in real bars
  – Understand what real humans do (bartenders, customers)
  – Design algorithms for the robot that operate in a similar manner

• Hardware and software
  – Reuse existing hardware (cat head, industrial arms, etc.)
  – Build modern, redistributable software that runs on multiple platforms

• Distributed design
  – Global agreement about the aims and interfaces of the system
  – Local implementation of individual components

• Competing needs
  – Researchers want an experimental lab tool
  – Funders want a demonstration system
  – Public? Media?

JAMES system architecture

PART II
HIGH-LEVEL DECISION MAKING AND PLANNING
Why does JAMES act as it does?

Real World
Visual Processor
Speech Recogniser
Parser
State Manager
Planner/Execution Monitor
Output Planner
Talking-Head Controller
Robot Motion Planner

Humans as planning agents

• Humans perceive and manipulate their environments by sensing, reasoning, and acting.
• E.g., planning a trip, preparing a meal, repairing a car, etc.
• In order to achieve particular goals in the world, a human must often reason about:
  – The actions it must perform, and
  – The order it needs to perform those actions.
• A human may also reason about the information it requires to perform an action, the objects affected by an action, how long an action takes, how much it costs to perform an action, etc.
⇒ Humans are pretty good at many complex types of planning tasks.

Decision making in JAMES

• What action should the robot perform next?
• There are many different design choices we can make to address this problem:
  – Hardcode the choice of action at each state.
  – Learn a policy that encodes the choice of action at each state.
⇒ Search for a plan that achieves the goals of the system by chaining together actions and reasoning about their effects.
• We use automated planning techniques from the AI community.
⇒ Many design choices still to make!

Robots as planning agents

• Robots must also perceive and manipulate their environments by sensing, reasoning, and acting.
• E.g., moving between rooms in a building, grasping objects, communicating with other agents, etc.
• Planning as a general computational process involves many related subproblems:
  – Observation
  – Representation
  – Reasoning
  – Execution
  – Recovery
  – Learning
  – ...
⇒ A hard computational problem, especially in real-world robot domains.
Task-based social interaction

Two people, A and B, each individually approach a bartender.

Bartender (to A):
How can I help you?
Person A: A pint of cider, please.

Person C approaches the bartender and attracts his attention.
Bartender (nods at A, then to C):
Just a moment please.

Bartender:
(Serves A)

Bartender (to B):
What will you have?
Person B:
A glass of red wine.

Bartender (nods at B):
(Serves B)

Bartender (to C):
Thanks for waiting.
How can I help you?
Person C:
I'd like a pint of bitter.

Bartender (nods at C):
(Serves C)

• Application: use general-purpose planning techniques to generate plans for controlling the bartender in a similar drink-ordering scenario.

Automated planning

• Automated planning techniques are good at building goal-directed plans of action under many challenging conditions, given a suitable description of a domain.

• A planning problem consists of:
  1. A representation of the properties and objects in the world and/or the agent's knowledge, usually described in a logical language,
  2. A set of state transforming actions,
  3. A description of the initial world/knowledge state,
  4. A set of goal conditions to be achieved.

• A plan is a sequence of actions that when applied to the initial state transforms the state in such a way that the resulting state satisfies the goal conditions.

Planning as a research field

• Classical planning
• Planning with incomplete information and sensing
• Planning under uncertainty
• Hierarchical planning
• Probabilistic planning
• Planning with costs and preferences
• Cost optimisation
• Temporal planning
• Planning with control knowledge
• SAT planning
• Heuristic search
• Plan execution
• ...

Example: classical planning

• The classical planning problem makes certain simplifying assumptions, e.g., complete knowledge of the world and deterministic actions.

• A (world) state is represented by a simple database of facts about the world. We can check if a fact is true in a state by querying the database (using closed-world reasoning and negation as failure).

• Actions are the sole means of change in the world.

• An action's preconditions specify the conditions that must be true in a state before an action can be applied (qualification problem).

• An action's effects specify the changes the action makes to a state.
  – Add list: facts an action makes true – added to the state,
  – Delete list: facts an action makes false – deleted from the state,
  – All other facts are unchanged (frame problem)
Example: classical planning (2)

<table>
<thead>
<tr>
<th>Action</th>
<th>Preconditions</th>
<th>Add list</th>
<th>Delete list</th>
</tr>
</thead>
<tbody>
<tr>
<td>pickup(x)</td>
<td>handEmpty</td>
<td>holding(x)</td>
<td>handEmpty</td>
</tr>
<tr>
<td>dropInBox(x,y)</td>
<td>onTable(x)</td>
<td>inBox(x,y)</td>
<td>onTable(y)</td>
</tr>
<tr>
<td></td>
<td>holding(x)</td>
<td>holding(x)</td>
<td>holding(y)</td>
</tr>
<tr>
<td></td>
<td>empty(y)</td>
<td>empty(y)</td>
<td></td>
</tr>
</tbody>
</table>

- Action operators: pickup, dropInBox
- Action parameters: x, y
- Properties: handEmpty, onTable, ...
- Objects: b1, o1, ...

Example: classical planning (3)

- Actions are state transforming: applying the effects of a action to a database state updates the database to produce a new database (denoting a new state) resulting from the execution of the action.
- We can generate plans by chaining together actions.
- E.g., one plan that achieves a state where inBox(o1, b1) holds is the action sequence: [pickup(o1), dropInBox(o1, b1)].
- Plans are often generated by searching through the set of possible states that can arise.

Example: classical planning (4)

- Problem: the assumptions of classical planning are not always realistic when an agent may have incomplete knowledge about the world, e.g.,
  - A robot with sensors exploring an unknown building,
  - A software agent in an operating system domain,
  - Agents interacting with other agents using natural language.
- The effects of physical actions not only change the state of the world, but also the mental state of the agent performing the action.
- Actions may have nondeterministic effects (e.g., sensing actions).
- ⇒ The problem often requires representing and reasoning about the planner’s knowledge and/or belief.
- ⇒ The term open world planning is often used to refer to planning situations where an agent doesn’t have complete information. The terms “planning with incomplete information and sensing”, “belief space planning”, “planning with knowledge”, among others, are also used.
Representing an agent's knowledge

- How can we represent an agent's incomplete knowledge about the state of the world for planning?
- Issues to consider:
  - What types of knowledge should be represented? Restrictions?
  - How do we represent the effects of sensors?
  - Does this representation enable practical plan generation?
  - Many approaches in the planning and knowledge representation communities.

Planning with Knowledge and Sensing

- One approach: Planning with Knowledge and Sensing (PKS), a "knowledge-level" conditional planner that builds plans based on what an agent knows (Petrick and Bacchus 2002, 2004).
- PKS represents its knowledge by using a set of five databases, each of which is restricted to a particular type of knowledge: \( K_f, K_v, K_w, K_x, LCW \).
- The contents of the databases have a fixed formal translation to formulae in a modal logic of knowledge which formally defines the planner's knowledge state.
- Actions are defined in terms of the changes they make to the planner's knowledge state (i.e., the databases), rather than the world state.
- Planning: actions are state transforming and give rise to new database sets when applied.
- PKS has previously been applied to traditional planning benchmarks, robot systems, web services, and operating system applications.

What knowledge can PKS represent?

- Relational facts about the world
  - `handEmpty`, `inDir(gcc, /usr/bin)`, `¬rainy`, ...
- Functional information
  - `combo(safe) ≠ 23-42-12`, `parentDir(local) = usr`, ...
- Disjunctive information
  - "I know the combination is 23-42-12 or 12-42-23."
- Plan time knowledge that will be resolved at execution time
  - "After checking the thermometer I will come to know the temperature."
- Local closed world information
  - "I know what objects are in the box."
- ...

Example: actions and planning in PKS

<table>
<thead>
<tr>
<th>Action</th>
<th>Preconditions</th>
<th>Effects</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>drop(x)</code></td>
<td><code>K(holding(x))</code></td>
<td><code>del(K_f, holding(x))</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>add(K_f, onFloor(x))</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>add(K_f, dropped(x))</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>del(K_f, ¬broken(x))</code></td>
</tr>
<tr>
<td></td>
<td></td>
<td><code>K(fragile(x)) ⇒ add(K_f, broken(x))</code></td>
</tr>
<tr>
<td><code>inspect(y)</code></td>
<td></td>
<td><code>add(K_w, broken(y))</code></td>
</tr>
</tbody>
</table>

- PKS uses an extended database model to represent actions.
- New knowledge states are computed by forward search:
  - Evaluate action preconditions against a set of databases,
  - Apply action effects to produce a new set of databases.
The planning problem

- Input (fusion): sensor information from vision and speech.
- Output (fission): postprocessed actions to generate arm motions, head behaviour, and speech.
- Domain includes:
  - Physical actions (e.g., handing over a drink),
  - Sensing actions (e.g., asking for a drink order) → often correspond to dialogue acts, and
  - Social behaviour (e.g., acknowledgements, thanking a customer).
- Construct a plan for the goal of serving all agents seeking attention.

Initial states and state updates

- Many aspects of the state are dynamic and cannot be determined a priori: agents in the scene, agents seeking attention, initial utterances.
- A state manager supports the planner by turning the continuous stream of noisy low-level sensor inputs into a discrete state representation.
- Sensors: linguistic interpreter, vision, robot arms, speech synthesiser, talking head.
- State properties reported to the planner:
  - Basic agent properties: location, face/torso orientation,
  - Inferred agent properties: social state, drink requests,
  - System output state: bad automatic speech recognition.
A bartender domain

- Properties
  - seeksAttn(\textit{a}) \quad \text{agent } \textit{a} \text{ seeks attention}
  - greeted(\textit{a}) \quad \text{agent } \textit{a} \text{ has been greeted}
  - ordered(\textit{a}) \quad \text{agent } \textit{a} \text{ has ordered}
  - transEnd(\textit{a}) \quad \text{the transaction with } \textit{a} \text{ has ended.}
  - inTrans = ?a \quad \text{the robot is interacting with } \textit{a}
  - request(\textit{a}) = ?d \quad \text{agent } \textit{a} \text{ has requested drink}

- Actions
  - greet(\textit{a}) \quad \text{greet an agent } \textit{a}
  - ask-drink(\textit{a}) \quad \text{ask agent } \textit{a} \text{ for a drink order}
  - ack-order(\textit{a}) \quad \text{acknowledge agent } \textit{a} \text{'s drink order}
  - serve(\textit{a},?d) \quad \text{serve drink } ?d \text{ to agent } \textit{a}
  - wait(\textit{a}) \quad \text{tell agent } \textit{a} \text{ to wait}
  - ack-wait(\textit{a}) \quad \text{thank agent } \textit{a} \text{ for waiting}
  - not-understand(\textit{a}) \quad \text{alert agent } \textit{a} \text{ that it was not understood}
  - bye(\textit{a}) \quad \text{end an interaction with agent } \textit{a}

 ⇒ Inspired by data collected from customers in the human studies.
 ⇒ No differentiation between different types of actions—it's all action!

Example: knowledge-level planning actions

- action ask-drink(\textit{a} : agent)
  - preconds: K(inTrans = ?a) & !K(ordered(?a))
  - !K(otherAttnReq) & !K(badASR(?a))
  - effects: add(Kf,ordered(?a)),
  - add(Kv,request(?a))

- action serve(\textit{a} : agent, ?d : drink)
  - preconds: K(inTrans = ?a) & K(ordered(?a)) &
  - Kv(request(?a)) & K(request(?a) = ?d)
  - !K(otherAttnReq) & !K(badASR(?a)) &
  - K(ackOrder(?a))
  - effects: add(Kf,served(?a))

A single customer plan

greet(a1), [Greet agent a1]
ask-drink(a1), [Ask a1 for drink order]
ack-order(a1), [Acknowledge a1’s order]
serve(a1,request(a1)), [Give the drink to a1]
bye(a1). [End the transaction]

- Simplest possible plan in the single agent case.
- Plans represent best-case scenarios based on current information and are subject to replanning if the world changes.

Action: ask-drink(a1)

Plan step

<output>
  <gesture type="Smile"/>
  </gesture-list>
  <speech-list>
    <speech type="query" politeness="4">
      <person id="A3":>
        <object type="offer">
          <object type="drink"/>
        </object>
      </person>
    </speech-list>
  </speech-list>
</output>

Multimodal output specification

State update

add(Kf, request(A1)=water))

Parsed speech input

<if>
  <node id="c:drink" pred="water" wmd="del" num="sg"/>
</if>
Action: serve(a1, request(a1))

A plan for two customers

wait(a2), [Tell agent a2 to wait]
greet(a1), [Greet agent a1]
ask-drink(a1), [Ask a1 for drink order]
ack-order(a1), [Acknowledge a1’s order]
serve(a1, request(a1)), [Give the drink to a1]
bye(a1), [End a1’s transaction]
ack-wait(a2), [Thank a2 for waiting]
ask-drink(a2), [Ask a2 for drink order]
ack-order(a2), [Acknowledge a2’s order]
serve(a2, request(a2)), [Give the drink to a2]
bye(a2). [End a2’s transaction]

• Each agent interaction is similar to the single agent plan.
• The plan additionally includes actions to manage the interaction order.

A single customer conditional plan

greet(a1), [Greet agent a1]
ask-drink(a1), [Ask a1 for drink order]
branch(request(a1)) [Form branching plan]
  K(request(a1)=juice): [If order is juice]
    serve(a1,juice) [Serve juice to a1]
  K(request(a1)=water): [If order is water]
    serve(a1,water) [Serve water to a1]
  K(request(a1)=beer): [If order is beer]
    serve(a1,beer) [Serve beer to a1]
bye(a1). [End the transaction]

• Branches let the planner consider order-specific actions/subdialogues.

Replanning when things go wrong

• Interactions with humans are inherently noisy. Plan execution is continually monitored to detect problems that may trigger replanning.
• Low-confidence speech recognition / timeouts
  ...
  ask-drink(a1) [Ask a1 for drink order]
  ??? [a1 was not understood]
  [Replan]
  not-understand(a1) [Alert a1 not understood]
  ask-drink(a1) [Ask a1 again for drink order]
  ...
  serve(a1, request(a1)) [Serve a1 their drink]
  bye(a1). [End the transaction]

• Overanswering
  greet(a1) [Greet a1]
  ??? [a1 says “I’d like a beer”]
  [Replan]
  serve(a1, request(a1)) [Serve a1 their drink]
  bye(a1). [End the transaction with a1]
### A more complex interaction

- `wait(A3,G1)`: Tell G2 to wait (with a nod)
- `greet(A1,G1)`: Greet group G1
- `ask-drink(A1,G1)`: Ask A1 for drink order
- `ack-order(A1,G1)`: Acknowledge A1's order
- `ask-drink(A2,G1)`: Ask A2 for drink order
- `ack-order(A2,G1)`: Acknowledge A2's order
- `serve(A1, request(A1,G1))`: Give the drink to A1
- `serve(A2, request(A2,G2))`: Give the drink to A2
- `bye(A2,G1)`: End G1's transaction
- `ack-wait(A3,G2)`: Acknowledge G2's wait
- `ask-drink(A3,G2)`: Ask A3 for drink order
- `ack-order(A3,G2)`: Acknowledge A3's order
- `serve(A3, request(A3,G3), G5)`: Give the drink to A3
- `bye(A3,G2)`: End G2's transaction

---

### Experimental results

- **Planning time** is typically quite short (≤0.1s), which doesn't impact the system's reaction time.
  - Less than 2s is usually okay.
  - Robot motions are slow.
  - Frequent replanning.

- **Study 1**: System tested with 2 customers at a time in a drink ordering scenario (31 participants × 3 interactions each), 95% success rate on delivering correct drinks.

- **Study 2**: More complex scenario (3 customers at a time, 40 participants), group detector, task only vs. social domain, 87% success rate.

- **Study 3**: Expected later this year.

---

### Design, human-robot interaction, and JAMES

- **Design decisions** at every stage of the project and all levels of software development have shaped the resulting JAMES system.
- There's no right answer to the design problem. Design choices are often motivated by decisions about (practical) tradeoffs.
- Social interaction places additional requirements on the design of a robot system: achieving a task goal isn’t always enough.
- The application of AI planning techniques in this context offers an alternative to mainstream approaches to interaction management.
- Human-robot interaction is a challenging (and rewarding) research field: there are always surprises and unexpected outcomes when working in the real world!
The other JAMES robot

• **Open research question:** The same high-level decision making components will be used on this robot platform. What difference will this embodiment make (if any)?

References


For more information on the JAMES Project visit [http://james-project.eu/](http://james-project.eu/).