Introduction to robotics

Robots: – this word has several very different meanings.

1. industrial robots: mechanical designs intended to perform a standard narrow set of pre-defined operations in a non-changing, carefully controlled environment;
2. remotely controlled robots: vehicles operated remotely by a human to work in dangerous or unaccessible environments;
3. miniature robots: microscopic electronic devices used in surgery to explore, measure or repair parts of human body;
4. mobile robots: relatively autonomous mechanical systems capable of functioning in the real world without direct human intervention;
5. software agents (softbots): programs operating relatively independently in computer environments.

AI is mostly associated with the last 2 types of robots: mobile robots and software robots.

Both these two types can be programmed to have relatively independent and goal-directed behavior.

AI researchers are interested to understand the principles of designing computer systems that can function independently in partially unknown and uncertain environments and can demonstrate purposeful behavior in the real world.

The AI hypothesis.

Recall that we considered the hypothesis that has 2 parts.

Part 1.

Our cognitive abilities may be accounted for in terms of symbolic manipulations over a rich collection of represented propositions.

We use rules of arithmetic to work with numbers (square root procedure).
We use rules of logic to reason about given logical statements.

Part 2.

To design an intelligent system we should

- represent what the system needs to know as symbolic sentences:
  We need a large Web of beliefs. Beliefs share common predicates.
- let the system to use rules of logic to reason what actions it can do to reach its goals.
- let the system do actions in the real world based on the conclusions it was able to derive.

Knowledge is power (F.Bacon).

The original AI approach to robotics is based on the idea that goal-directed behavior of robots (and other agents) is derived from

- their **declarative knowledge** about the world and
- from the **procedural knowledge** of how to build certain stereotypical complex behaviors from given primitive actions.

This approach follows a long standing philosophical tradition.

The first implemented mobile robot based on the original AI approach:
Shakey, Stanford Research Institute, 1968–1973.

Shakey was not successful for several reasons:

logical reasoning was implemented inefficiently, because it was not known at that time how to do it efficiently;

there was no clear separation between responsibilities of logical, high-level control module and responsibilities of low-level control modules (real-time obstacle avoidance);

planning required unbounded search.

Behavior-based robotics

There are other approaches pursued in several universities.

The most well-known alternative to previously mentioned AI approach has been proposed by Rodney Brooks (MIT):

- Slogan 1: "The world is its own best model".

  Denies the need for internal representation of the world.

  No model is required, in particular no logical model is used.

  But because the system has no model, it cannot plan and it cannot reason.

- Slogan 2: "Intelligence without reasoning".

  A complex behavior is the result of an interaction of a robot with the complex world, but the structure of control system can be simple.

- Slogan 3: "Elephants do not play chess".

  A complex behavior is an hierarchy of simple behaviors, like obstacle-avoidance, wall-following, wandering

This research direction is mainly successful in

– the design of artificial insects: biologists are sometimes interested to see whether it is possible to build a mechanical system with motor coordination comparable with insects (e.g., cockroach)

– the study of simple forms of collective behavior (flocking)

– the design of so called "humanoid robots": complex mechanical systems that resemble parts of human body.

A mechanical design resembling human torso can
coordinate movements of hands;
a mechanical design resembling legs can
climb stairs and walk;
a mechanical design resembling human face can simulate facial
expressions associated usually with certain emotions.

If you are curious to see these systems, follow links to robotics video mentioned on our Web page.

All these very intricate and interesting mechanical systems are not able to demonstrate a purposeful behavior: they have no internal representation of the world and for this reason cannot deliberate about achieving goals.

A logic-based approach.

• Knowledge is represented by certain symbols (e.g., predicates).

• The robot is given an explicit logical model of the world supplemented by several other special purpose models. For example, the robot can also use a map of the corridor as a geometrical model of an office environment.

• Once each statement is expressed by a predicate, reasoning is performed by means of certain rules which allow the system to derive conclusions from a given set of statements.

• This approach can be implemented in Prolog, because Prolog provides predicates and rules which can be used to represent knowledge and to do logical reasoning.

• A logical program responsible for reasoning must be interfaced with low-level software. Low-level software can consist of several modules responsible for motion of the robot, taking into account sensory information, avoiding collisons with obstacles, computing the current location of the robot, etc.

Planning in logic.

Before a robot starts to do something intelligent in the real world, it has to plan what to do. In other words, it has to deliberate what decisions to make and in what sequence.

We considered a very simplified version of the planning task in class.

There are

- an initial state
- a goal state
- legal moves from one state to another reachable state.

The planning task is how to find a sequence of possible actions that leads from the initial state to the goal state.

We considered planning as a reasoning task (and its implementation in Prolog) because reasoning provides

1. the most general mechanism for planning;
2. a flexible (elaboration tolerant) mechanism for planning.

Planning in Prolog.

We have seen only simple examples when the planning task can be solved in Prolog by relatively short sequences of actions.

Situations and fluents formulation of the planning problem.

```
solve_problem(L) :- reachable(S,L), goal_state(S).

reachable(S,[ ]) :- initial_state(S).

reachable(S2, [A | L]) :- reachable(S1, L), legal_move(S2,A,S1),
                     not useless(A,L).

initial_state([ ]).
legal_move([A | S], A, S) :- poss(A,S).
```

What this program does to find a plan? It searches exhaustively through all lists L of possible actions.

The longer is a desired sequence of actions, the more time this search will take. If a shortest plan that solves the problem contains 10 or 15 actions, it may take hours before the system will find a plan.

Reminder: the predicate *useless(A,L)* helps to cut wrong branches of the search tree when our generic problem solver looks for a list of actions leading to the goal state.

Rules that define the predicate *useless(A,L)* express domain specific knowledge, but it is not always easy to formulate these rules.

Without carefully designed rules for the predicate *useless(A,L)* the planning task that has a solution with 20 actions or more is computationally infeasible.

So, what else we can do to compute long plans ?

Sometimes given a planning problem, a programmer can write a high-level program that will guide the search (or, if a problem is *'well-structured'*, eliminate the search).

This is a feasible alternative if a person who writes a program has a good procedural intuition on how to reach the goal state from an initial state.

This program can take non-deterministically one of "either-or" alternatives and can choose non-deterministically a value of x that makes a certain condition true.

Hence, the search will concentrate only on choosing one of these non-deterministic alternatives.

Golog

Using programming constructs **if-then-else**, **while-do**, **either-or**, **choose**, tests, recursive procedures and some other constructs one can write programs that will spend less time on search. Those programs will help to find plans composed from given primitive actions.

Golog is a high-level logic-based programming language designed to guide the search.

The standard planning task can be expressed in Golog as a repeated non-deterministic choice between all possible primitive actions until the goal is achieved.

All other planning algorithms can also be expressed in Golog: it is a general purpose language for solving planning problems.

Additional advantage: using successor-state axioms we can compute values of fluents after each action, i.e. we can update the current logical model of the world.

Additional complications.

In the real world, an action may have several different results, depending on circumstances that we cannot control.

In other words, actions are not necessarily deterministic: they often have several different outcomes with associated probabilities.

Both low-level actions (locomotion) and high-level actions (what to do next) can have probabilistic results.

When we deliberate we can think that if the result of this action will be this, I'll do the second action, but if the result will turn out to be different, then I'll do the third action.

In other words, we must consider conditional plans, not just simple sequences of actions.

Additional complications: cont.

In addition, it may often be convenient to think about probabilistic results of actions in terms of costs/rewards that we obtain when actions yield certain outcomes.

Then different conditional plans will result in different total accumulated rewards.

When we consider a conditional plan, by computing the average over all possible outcomes, we can compute the expected total reward for doing this plan.

Then the planning task is more realistically can be described as finding an optimal (on average) conditional plan.

Sometimes, this task is formulated as the problem of making an optimal (on average) sequence of decisions.

Computing optimal conditional plans.

Intuitively, we feel that the task of finding an optimal conditional plan is even more complex than the task of finding a plan when all actions are deterministic.

Hence, in realistic scenarios it can be infeasible for a robot to compute an optimal conditional plan in real time.

From the other hand, it is not possible to program the robot directly either: the programmer cannot find himself/herself what the robot has to do to behave optimally.

Can we use Golog?

Yes, it is possible to write natural Golog programs that will guide the search for an optimal conditional plan.

1) The Golog programmer will write precondition and successor state axioms for probabilistic actions.

2) The Golog programmer will specify reward functions for results of actions and estimates of probabilities of different outcomes for each action.

3) The Golog programmer will specify what should be done without deliberation and where the robot has to deliberate to make optimal decisions that take into account all possible circumstances.