Issues for Single Robot Control

Dr. Daisy Tang

All lecture notes are adapted from Dr. Lynne Parker’s lecture notes on Software for Intelligent Robotics
Outline

- Introduction to Robots
- Practical robot implementation issues
- Introduction to robot behaviors
- Player/Stage simulator
What is Intelligent Robot?

- A mechanical creature which can function autonomously

- A physical agent that performs tasks by interacting with the physical world

- A physical agent that generates “intelligent” connection between perception and action
Intelligent Robot (cont.)

- **Intelligent**: robot does not do things in a mindless, repetitive way

- **Mechanical creature**: mechanical building blocks instead of biological parts; different from a computer

- **Function autonomously**: the robot can operate, self-contained, under all reasonable conditions without help from human operator
Fiction vs. Real-World Fact

- Hollywood robots:
  - human-like capabilities
  - sense all, know all

- Real-world robots:
  - insect or simple animal capabilities
  - sense little, know little

Star wars robots
I Robot
Wall-E
Hospital delivery robots
Pool cleaning robot
Current-State-Of-The-Art Robots
What Composes a Robot?

- Sensors

- Effectors and actuators
  - locomotion and manipulation

- Controllers
  - coordinate information from sensors with commands for the robot’s actuators

- Robot = an autonomous system which exists in the physical world, can sense its environment and can act on it to achieve some goals
Basic Robot Software Issues

- How do you perceive?
- How do you control?
- How do you act?

Environment

Perception → Control → Action

- sense/detect
- behave, react, reason, ...
- through effectors: wheels, legs, ...

Flowchart showing the relationship between perception, control, and action in a robot system.
In Robot Design Choices, Must Consider Real-World Challenges

- **Autonomous**: robot makes majority of decisions on its own; no human-in-the-loop control
- **Mobile**: robot does not have fixed bases
- **Unstructured**: environment has not been specifically designed to make robot’s job easier
- **Dynamic**: environment may change unexpectedly
- **Partially observable**: robot cannot sense entire state of world
- **Uncertain**: sensor readings are noisy; effector output is noisy
Practical Robot Implementation Issues
Behavior-Based / Reactive

Philosophy:

“World is its own best model; therefore, don’t try to build another world model”

More recent robot projects use this approach, with better success.
Typical Mobile Robot Implementation Architecture

- Essentially: PC on wheels/legs/tracks/...
The Way You Structure Your Code

- Two options:
  - Multi-threaded program
  - Single process with “operating system”-like time-slicing of procedures

- Both are useful
- We focus on a single process
Example: Follow Wall and Obey Commands

- Assume we have the following functions needed:
  - Communications to operator interface
  - Sonar to follow wall and avoid obstacles
Typical “Single Process” Control To Achieve Parallelism

```c
int wall_follower() {/* one time slice of work */}
int obstacle_avoider() {/* one time slice of work */}
int communicator() {/* one time slice of work */}
int controller_arbitrator() {/* decides what action to take */}

main( )
{
    while (1) {
        wall_follower();
        obstacle_avoider();
        communicator();
        controller_arbitrator();
    }
}
```

Dependent upon programmer to ensure individual functions return after “time slice” completed!
PlayerClient robot("localhost");
Position2dProxy pp(&robot, 0);
...

while (1) {
    ...
    pp.SetSpeed(speed, turnrate);
    ...
}

Robot continues to execute the given velocity command until another setspeed command issued.

Thus, duration of “time slice” is important.
Challenges for Software for Intelligent Robots

- Software issues enabling autonomous mobile robots to accomplish given objectives in unstructured, dynamic, partially observable, and uncertain environments
Examples of Unstructured Environment

- **Examples of unstructured environment:**
  - Nearly all natural (non-man-made) environments:
    - deserts, forests, fields
  - To some extent, man-made environments not specifically designed for robots

- **Impact:**
  - Difficult to make assumptions about sensing expectations
  - Difficult to make assumptions about environmental characteristics

- **Example applications:**
  - space robot, fire fighting, military ...
Sources and Effect of Dynamic Environment

Sources of dynamic environment:

- Other robots/agents in the area
  - Teammates
  - Adversaries
  - Neutral actors
- Natural events (e.g., rain, smoke, haze, moving sun, power outages, etc.)

Impact:

- Assumptions at beginning of mission may become invalid
- Sensing/Action loop must be tight enough that environment changes don’t invalidate decisions
Causes and Effect of Partially Observable Environment

- Causes of partially observable environment:
  - limited resolution sensors
  - reflection, occlusion, absorption

Example:
Glass walls--laser sensors tricked
Sources and Effect of Uncertainty/Noise

- **Sources of sensor noise:**
  - limited resolution sensors
  - sensor reflection, absorption
  - poor quality sensor conditions (e.g., low lighting for cameras)

- **Sources of effector noise:**
  - friction: constant or varying (e.g., carpet vs. tile; clean vs. dirty floor)
  - Slippage (e.g., when turning or on dusty surface)
  - Varying battery level (drainage during mission)

- **Impact:**
  - Sensors difficult to interpret
  - Same action has different effects when repeated
  - Incomplete information for decision making
Example of Effect of Dynamic Environment

Possible control code:
while (forever) do:
{
  free = check_for_obstacle_to_goal();
  if (free)
    move_straight_to_goal();
  sleep(a_while);
}  briefly
"Exact" Motions vs. Servo Loops

Two possible control strategies:

- **"Exact" motions:**
  - Turn right by amount $\theta$
  - Go forward by amount $d$

- **Servo loop:**
  - if to the left of desired trajectory, turn right
  - if to the right of desired trajectory, turn left
  - if on line with desired trajectory, go straight
  - if error to desired trajectory is large, go slow
  - if error to desired trajectory is small, go fast
Effect of Noise for Exact Control Method

“Exact” method:

Current robot position & orientation

$\theta_1$, $d_1$: actual angle, distance traveled;

Noise $\Rightarrow$ overshoot goal; have to turn back to goal

Doesn’t generate good performance
Effect of Noise for Servo Method

- Servo method:

Much better performance in presence of noise.
Introduction to Behaviors
What is “Behavior”?

- Scientists believe the fundamental building block of natural intelligence is a behavior.

- **Behavior**: mapping of sensory inputs to a pattern of motor actions that are used to achieve a task.

- Behaviors serve as the **basic building blocks for the robotic software**.
Three Broad Categories of Behaviors

- **Reflexive** behaviors:
  - stimulus-response (S-R)
  - hard-wired for fast response; neural circuits ensure stimulus is directly connected to the response
  - example: (physical) knee-jerk reaction

- **Reactive** behaviors:
  - learned, but executed without conscious thought
  - example: muscle memory, playing piano, riding bicycle, running, etc.

- **Conscious** behaviors:
  - required deliberative thought
  - example: writing computer code, completing your tax return, etc.

- The reactive or behavior-based paradigm will make extensive use of reflexive behaviors
Perception in Behaviors

Behaviors depend on perception

Two functions of perception:
- **Release**: to release a behavior
- **Guide**: to provide information needed to accomplish a behavior

**Action-oriented perception:**
- perception filters the incoming sensory stream to extract information specific to the task at hand

**Affordance**: “perceivable potentialities of the environment for an action”
- Example: Color “red” to a baby arctic tern is perceivable, and represents the potential for feeding
- Can also guide a behavior
Schema Theory

- **Schema theory**
  - Provides a helpful way of programming behaviors into an object-oriented format

- **Schema**
  - Consists of:
    - Information on how to act and/or perceive (knowledge, data structures, models)
    - Computational process by which it achieves the activity (algorithm)
  - Is a generic template for how to do some activity
Behaviors and Schema Theory

Releaser

Sensory Input → BEHAVIOR → Pattern of Motor Actions

Perceptual Schema  Motor Schema

sensing  physical activity
Example of Toad’s Feeding Behavior Using Schema

Releaser: Appearance of small, moving object

Sensory Input: Toad’s vision

FEEDING BEHAVIOR

Pattern of Motor Actions: Toad’s legs

Perceptual Schema:
get coordinates of small, moving object

Motor Schema:
turn to coordinates of small, moving object
Perception-Action Schema Relationships

Environmental Sensors

Motor Schemas

Environmental Sensor 1 (ES1)

Motor Schema 1 (MS1)

Perceptual Schema 1 (PS1)

Perceptual Subschema 1 (PSS1)

Perceptual Schema 3 (PS3)

Environmental Sensor 2 (ES2)

Perceptual Schema 2 (PS2)

Environmental Sensor 3 (ES3)

Perceptual Schema 3 (PS3)

Perceptual Subschema 2 (PSS2)

Motor Schema 2 (MS2)

Vector (\(\Sigma\))

Robot

Motors

PS = Perceptual Schema
PSS = Perceptual Subschema
MS = Motor Schema
ES = Environmental Sensor
Output of Motor Schemas Defined as Vectors

- **Output Vector**: consists of both orientation and magnitude components
  - $V_{\text{magnitude}}$ denotes magnitude of resultant response vector
  - $V_{\text{direction}}$ denotes orientation

Length of arrow = $m$ = magnitude
Angle of arrow = $d$ = direction
Potential Fields

- Potential field: array (or field) of vectors representing space
- Vector space is 2D world, like bird’s eye view of map
- Map divided into squares, creating (x, y) grid
- Each element represents square of space
- Perceivable objects in world exert a force field on surrounding space
Five Primitive Potential Fields

Uniform

Perpendicular

Attraction

Repulsion

Tangential
Magnitude Profiles

- Change in velocity in different parts of the field

Field closer to an attractor/repellor will be stronger
Programming a Single Potential Field

- Repulsive field with linear drop-off:

\[ V_{\text{direction}} = 180^\circ \]

\[ V_{\text{magnitude}} = \begin{cases} 
\frac{(D-d)}{D} & \text{for } d \leq D \\
0 & \text{for } d > D 
\end{cases} \]

where \( D \) is max range of field’s effect
Pseudocode for Single Repulsive Field

typedef struct {
    double magnitude;
    double direction;
} vector;

vector repulsive(double d, double D)
{
    if (d <= D)
    {
        outputVector.\_direction = -180;  // turn around
        outputVector.\_magnitude = (D-d)/D;  // linear dropoff
    }
    else
    {
        outputVector.\_direction = 0.0;
        outputVector.\_magnitude = 0.0;
    }
    return outputVector;
}
Important Note

- Entire field does not have to be computed

- Only portion of field affecting robot is computed

- Robot uses functions defining potential fields as its position to calculate component vector
More Motor Schema Encodings

- **Move-to-goal:**
  - \( V_{\text{magnitude}} \) = fixed gain value
  - \( V_{\text{direction}} \) = towards perceived goal

- **Move-ahead:**
  - \( V_{\text{magnitude}} \) = fixed gain value
  - \( V_{\text{direction}} \) = specified compass direction

- **Noise:**
  - \( V_{\text{magnitude}} \) = fixed gain value
  - \( V_{\text{direction}} \) = random direction changed every \( p \) time steps
Combing Fields/Behaviors

- Compute each behavior’s potential field
- Sum vectors at robot’s position to get resultant output vector

\[ R = \Sigma (G_i \cdot R_i) \]

- Issues:
  - local minima: vectors may sum to zero

- Solutions:
  - avoid past behavior
  - inject noise
A Case Study

- Trash collecting robot
Player/Stage
Introduction to Player/Stage

- The Player Stage Project

- Installation guide + Quick start
  - [http://www.cpp.edu/~ftang/courses/player-stage/player-stage-intro.htm](http://www.cpp.edu/~ftang/courses/player-stage/player-stage-intro.htm)
What is Player?

Player is a network server for robot control. It provides a simple interface to the robot’s sensors and actuators over the IP network.
Your Readings

- Player manual
  - [http://playerstage.sourceforge.net/doc/Player-2.1.0/player/index.html](http://playerstage.sourceforge.net/doc/Player-2.1.0/player/index.html)
- Installation (optional)
- Quick start
- Client libraries (write your own control program)
- Be able to write/use configuration files
Exploration

- Try different sample programs
- Make some changes and see what happens
- Pick out some sample programs, look up the function definitions, and understand what they are doing
Notice

- Project 1 assigned.