Distributed Intelligent Systems – W4
An Introduction to Genetic Algorithms and Particle Swarm Optimization and their Application to Single-Robot Control Shaping
Outline

• Machine-learning-based methods
  – Rationale for embedded systems
  – Classification and terminology
• Genetic Algorithms (GA)
• Particle Swarm Optimization (PSO)
• Comparison between GA and PSO
• Noise-resistance
• Application to single-robot control shaping
  – Obstacle avoidance
  – Homing
  – Co-evolution hardware and software
Rationale and Classification
Why Machine-Learning?

- Complementarity to a model-based/engineering approaches: when low-level details matter (optimization) and/or good models do not exist (design)!

- When the design/optimization space is too big (infinite)/too computationally expensive (e.g. NP-hard) to be systematically searched

- Automatic design and optimization techniques

- Role of engineer reduced to specifying performance requirements and problem encoding
Why Machine-Learning?

• There are design and optimization techniques robust to noise, nonlineararities, discontinuities
• Individual real-time adaptation to new environmental conditions; i.e. increased individual flexibility when environmental conditions are not known/cannot predicted a priori
• Search space: parameters and/or rules
ML Techniques: Classification Axis 1

- Supervised learning: off-line, a teacher is available
- Unsupervised learning: off-line, teacher not available
- Reinforcement-based (or evaluative) learning: on-line, no pre-established training and evaluation data sets
Supervised Learning

- Off-line
- Training and test data are separated, a teacher is available
- Typical scenario: a set of input-output examples is provided to the system, performance error given by difference between system output and true/teacher-defined output, error fed to the system using optimization algorithm so that performance is increased over trial
- The generality of the system after training is tested on examples not previously presented to the system (i.e. a “test set” exclusive from the “training set”)
Unsupervised Learning

– Off-line
– No teacher available, no distinction between training and test data sets
– Goal: structure extraction from the data set
– Examples: data clustering, Principal Component Analysis (PCA) and Independent Component analysis (ICA)
Reinforcement-based (or Evaluative) Learning

- On-line
- No pre-established training or test data sets
- The system judges its performance according to a given metric (e.g., fitness function, objective function, performance, reinforcement) to be optimized
- The metric does not refer to any specific input-to-output mapping
- The system tries out possible design solutions, does mistakes, and tries to learn from its mistakes
ML Techniques: Classification Axis 2

- **In simulation**: reproduces the real scenario in simulation and applies there machine-learning techniques; the learned solutions are then downloaded onto real hardware when certain criteria are met.

- **Hybrid**: most of the time in simulation (e.g. 90%), last period (e.g. 10%) of the learning process on real hardware.

- **Hardware-in-the-loop**: from the beginning on real hardware (no simulation). Depending on the algorithm more or less rapid.
ML Techniques: Classification Axis 3

ML algorithms require sometimes fairly important computational resources (in particular for multi-agent algorithms), therefore a further classification is:

- **On-board**: machine-learning algorithm run on the system to be learned (no external unit)

- **Off-board**: the machine-learning algorithm runs off-board and the system to be learned just serves as embodied implementation of a candidate solution
Selected Evaluative Machine-Learning Techniques

- Evolutionary computation
  - Genetic Algorithms (GA) multi-agent, W4
  - Genetic Programming (GP) multi-agent
  - Evolutionary Strategies (ES) multi-agent

- Swarm Intelligence
  - Ant Colony Optimization (ACO) multi-agent, W2
  - Particle Swarm Optimization (PSO) multi-agent, W4

- Learning
  - In-Line Adaptive Learning single-agent, W6
  - Reinforcement Learning (RL) single-agent
Genetic Algorithms
Genetic Algorithms Inspiration

• In natural evolution, organisms adapt to their environments – better able to survive over time

• Aspects of evolution:
  – Survival of the fittest
  – Genetic combination in reproduction
  – Mutation

• Genetic Algorithms use evolutionary techniques to achieve parameter optimization and solution design
GA: Terminology

- **Population**: set of $m$ candidate solutions (e.g. $m = 100$); each candidate solution can also be considered as a genetic individual endowed with a single chromosome which in turn consists of multiple genes.

- **Generation**: new population after genetic operators have been applied ($n = \#$ generations e.g. 50, 100, 1000).

- **Fitness function**: measurement of the efficacy of each candidate solution

- **Evaluation span**: evaluation period of each candidate solution during a given generation. The time cost of the evaluation span differs greatly from scenario to scenario: it can be extremely cheap (e.g., simply computing the fitness function in a benchmark function) or involve an experimental period (e.g., evaluating the performance of a given control parameter set on a robot)

- **Life span**: number of generations a candidate solution survives

- **Population manager**: applies genetic operators to generate the candidate solutions of the new generation from the current one
Evolutionary Loop: Several Generations

Ex. of end criteria:
• # of generations
• best solution performance
• …
Generation Loop

- Population replenishing
- Crossover and mutation
- Selection
- Fitness measurement and encoding (phenotype → genotype)
- Decoding (genotype → phenotype)
- Evaluation of Individual Candidate Solutions

System

Population Manager
GA: Encoding & Decoding

phenotype $\xrightarrow{\text{encoding}}$ genotype $(\text{chromosome})$ $\xrightarrow{\text{decoding}}$ phenotype

- phenotype: usually represented by the whole system which can be evaluated; the whole system or a specific part of it (problem formulation done by the engineer) is represented by a vector of dimension $D$; vector components are usually real numbers in a bounded range.

- genotype: chromosome = string of genotypical segments, i.e. genes, or mathematically speaking, again a vector of real or binary numbers; vector dimension varies according to coding schema ($\geq D$); the algorithm search in this hyperspace.

$G_1, G_2, G_3, G_4, \ldots, G_n$

$G_i =$ gene $=$ binary or real number

Encoding: real-to-real or real-to-binary via Gray code (minimization of nonlinear jumping between phenotype and genotype)

Decoding: inverted operation

Rem:

- Artificial evolution: usually one-to-one mapping between phenotypic and genotypic space
- Natural evolution: 1 gene codes for several functions, 1 function coded by several genes.
GA: Basic Operators

- **Selection:** roulette wheel (selection probability determined by normalized fitness), ranked selection (selection probability determined by fitness order), elitist selection (highest fitness individuals always selected)
- **Crossover** (e.g. 1 point, \( p_{\text{crossover}} = 0.2 \))

![Crossover Diagram]

- **Mutation** (e.g. \( p_{\text{mutation}} = 0.05 \))

![Mutation Diagram]
Particle Swarm Optimization
Reynolds’ Rules for Flocking

1. **Separation**: avoid collisions with nearby flockmates

2. **Alignment**: attempt to match velocity (speed and direction) with nearby flockmates

3. **Cohesion**: attempt to stay close to nearby flockmates

More on Week 8
PSO: Terminology

- **Population**: set of candidate solutions tested in one time step, consists of $m$ particles (e.g., $m = 20$)

- **Particle**: represents a candidate solution; it is characterized by a velocity vector $v$ and a position vector $x$ in the hyperspace of dimension $D$

- **Evaluation span**: evaluation period of each candidate solution during one a time step; as in GA the evaluation span might take more or less time depending on the experimental scenario.

- **Fitness function**: measurement of efficacy of a given candidate solution during the evaluation span

- **Population manager**: update velocities and position for each particle according to the main PSO loop

- **Principles**: imitate, evaluate, compare
Evolutionary Loop: Several Generations

Ex. of end criteria:
• # of time steps
• best solution performance
• …
Initialization: Positions and Velocities
The Main PSO Loop – Parameters and Variables

- **Functions**
  - `rand()`: uniformly distributed random number in [0,1]

- **Parameters**
  - `w`: velocity inertia (positive scalar)
  - `c_p`: personal best coefficient/weight (positive scalar)
  - `c_n`: neighborhood best coefficient/weight (positive scalar)

- **Variables**
  - `x_{ij}(t)`: position of particle i in the j-th dimension at time step t (j = [1,D])
  - `v_{ij}(t)`: velocity particle i in the j-th dimension at time step t
  - `x_{ij}^*(t)`: position of particle i in the j-th dimension with maximal fitness up to iteration t
  - `x_{i'j}^*(t)`: position of particle i’ in the j-th dimension having achieved the maximal fitness up to iteration t in the neighborhood of particle i
The Main PSO Loop

At each time step $t$
  for each particle $i$
    for each component $j$

<table>
<thead>
<tr>
<th>update velocity</th>
<th>$v_{ij}(t+1) = wv_{ij}(t) + c_p \text{rand}()(x^<em><em>i - x</em>{ij}) + c_n \text{rand}()(x^</em><em>j - x</em>{ij})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>then move</td>
<td>$x_{ij}(t+1) = x_{ij}(t) + v_{ij}(t+1)$</td>
</tr>
</tbody>
</table>
The main PSO Loop
- Vector Visualization

Here I am!

My position for optimal fitness up to date

The position with optimal fitness of my neighbors up to date

$p$-best

$n$-best
Neighborhoods Types

- **Size:**
  - Neighborhood index considers also the particle itself in the counting
  - **Local:** only k neighbors considered over m particles in the population ($1 < k < m$); $k=1$ means no information from other particles used in velocity update
  - **Global:** m neighbors

- **Topology:**
  - Geographical
  - Social
  - Indexed
  - Random
  - ...
Neighborhood Examples: Geographical vs. Social
Neighborhood Example: Indexed and Circular (lbest)

Particle 1’s 3-neighbourhood

Virtual circle
PSO Animated Illustration

Global optimum

© M. Clerc
GA vs. PSO
### GA vs. PSO - Qualitative

<table>
<thead>
<tr>
<th>Parameter/function</th>
<th>GA</th>
<th>PSO</th>
</tr>
</thead>
<tbody>
<tr>
<td>General features</td>
<td>Multi-agent, probabilistic search</td>
<td>Multi-agent, probabilistic search</td>
</tr>
<tr>
<td>Individual memory</td>
<td>no</td>
<td>yes (randomized hill climbing)</td>
</tr>
<tr>
<td>Individual operators</td>
<td>mutation</td>
<td>personal best position history, velocity inertia</td>
</tr>
<tr>
<td>Social operators</td>
<td>selection, crossover</td>
<td>neighborhood best position history</td>
</tr>
<tr>
<td>Particle’s variables (tracked by the population manager)</td>
<td>position</td>
<td>position and velocity</td>
</tr>
</tbody>
</table>
## GA vs. PSO - Qualitative

<table>
<thead>
<tr>
<th>Parameter/function</th>
<th>GA</th>
<th>PSO</th>
</tr>
</thead>
<tbody>
<tr>
<td># of algorithmic parameters (basic)</td>
<td>$p_m, p_c, \text{ selection par.}$</td>
<td>$w, c_n, c_p, k, m, \text{ position and velocity range}$</td>
</tr>
<tr>
<td></td>
<td>(1), $m$, position range (1) = 5</td>
<td>(2) = 7</td>
</tr>
<tr>
<td>Population diversity</td>
<td>somehow tunable via $p_c/p_m$ ratio and selection schema</td>
<td>Mainly via local neighborhood</td>
</tr>
<tr>
<td>Global/local search balance</td>
<td>somehow tunable via $p_c/p_m$ ratio and selection schema</td>
<td>Tunable with $w$ ($w \uparrow \rightarrow$ global search; $w \downarrow \rightarrow$ local search)</td>
</tr>
<tr>
<td>Original search space</td>
<td>discrete (but continuous ok nowadays)</td>
<td>continuous (but some work on discrete nowadays)</td>
</tr>
</tbody>
</table>
GA vs. PSO - Quantitative

- Goal: minimization of a given $f(x)$
- Standard benchmark functions with thirty terms ($n = 30$) and a fixed number of iterations
- All $x_i$ constrained to $[-5.12, 5.12]$

<table>
<thead>
<tr>
<th>Function Name</th>
<th>Function</th>
<th>Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sphere</td>
<td>$f_1(\bar{x}) = \sum_{i=1}^{n} x_i^2$</td>
<td>400</td>
</tr>
<tr>
<td>Generalized</td>
<td>$f_2(\bar{x}) = \sum_{i=1}^{n-1} [100(x_i^2 - x_{i+1})^2 + (1 - x_i)^2]$</td>
<td>20000</td>
</tr>
<tr>
<td>Rosenbrock</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rastrigin</td>
<td>$f_3(\bar{x}) = \sum_{i=1}^{n} [x_i^2 - 10\cos(2\pi x_i) + 10]$</td>
<td>400</td>
</tr>
<tr>
<td>Griewank</td>
<td>$f_4(\bar{x}) = 1 + \frac{1}{4000} \sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right)$</td>
<td>400</td>
</tr>
</tbody>
</table>
GA vs. PSO - Quantitative

- GA: Roulette Wheel for selection, mutation applies numerical adjustment to gene
- PSO: lbest ring topology with neighborhood of size 3
- Algorithm parameters used (but not thoroughly optimized):

<table>
<thead>
<tr>
<th>GA</th>
<th>PSO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Population Size</td>
<td>Swarm Size</td>
</tr>
<tr>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>Crossover Probability</td>
<td>Personal Best Attraction</td>
</tr>
<tr>
<td>0.6</td>
<td>2.0</td>
</tr>
<tr>
<td>Mutation Probability</td>
<td>Neighborhood Best Attraction</td>
</tr>
<tr>
<td>0.05</td>
<td>2.0</td>
</tr>
<tr>
<td>Mutation Range</td>
<td>Inertia Factor</td>
</tr>
<tr>
<td>[-0.5, 0.5]</td>
<td>0.6</td>
</tr>
</tbody>
</table>
GA vs. PSO - Quantitative

**Bold: best results; 30 runs; no noise on the performance function**

<table>
<thead>
<tr>
<th>Function (no noise)</th>
<th>GA (mean ± std dev)</th>
<th>PSO (mean ± std dev)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sphere</td>
<td>0.02 ± 0.01</td>
<td>0.00 ± 0.00</td>
</tr>
<tr>
<td>Generalized Rosenbrock</td>
<td>34.6 ± 18.9</td>
<td>7.38 ± 3.27</td>
</tr>
<tr>
<td>Rastrigin</td>
<td>157 ± 21.8</td>
<td>48.3 ± 14.4</td>
</tr>
<tr>
<td>Griewank</td>
<td>0.01 ± 0.01</td>
<td>0.01 ± 0.03</td>
</tr>
</tbody>
</table>
GA vs. PSO – Overview

• According to most recent research, PSO outperforms GA on most (but not all!) continuous optimization problems
• GA still much more widely used in general research community and robust to continuous and discrete optimization problems
• Because of random aspects, very difficult to analyze either metaheuristic or make guarantees about performance
Design and Optimization of Obstacle Avoidance Behavior using Genetic Algorithms
Evolving a Neural Controller

**Note:** In our case we evolve synaptic weights but Hebbian rules for dynamic change of the weights, transfer function parameters, … can also be evolved (see Floreano’s course)
Evolving Obstacle Avoidance
(Floreano and Mondada 1996)

Defining performance (fitness function):

\[ \Phi = V(1 - \sqrt{\Delta V})(1 - i) \]

- \( V = \text{mean speed of wheels, } 0 \leq V \leq 1 \)
- \( \Delta v = \text{absolute algebraic difference between wheel speeds, } 0 \leq \Delta v \leq 1 \)
- \( i = \text{activation value of the sensor with the highest activity, } 0 \leq i \leq 1 \)

**Note:** Fitness accumulated during evaluation span, normalized over number of control loops (actions).
Shaping Robot Controllers

Note:
Controller architecture can be of any type but worth using GA/PSO if the number of parameters to be tuned is important.
Evolved Obstacle Avoidance Behavior

Generation 100, on-line, off-board (PC-hosted) evolution

Note: Direction of motion NOT encoded in the fitness function: GA automatically discovers asymmetry in the sensory system configuration (6 proximity sensors in the front and 2 in the back)
Evolving Obstacle Avoidance

Evolved path

Fitness evolution
Expensive Optimization and Noise Resistance
Expensive Optimization Problems

Two fundamental reasons making robot control design and optimization expensive in terms of time:

1. **Time for evaluation** of candidate solutions (e.g., tens of seconds) $\gg$ time for application of metaheuristic operators (e.g., milliseconds)

2. **Noisy performance evaluation** disrupts the adaptation process and require multiple evaluations for actual performance
   - Multiple evaluations at the same point in the search space yield different results
   - Noise causes decreased convergence speed and residual error
Algorithmic Principles for Dealing with Expensive Optimization Problems

- Better information about candidate solution can be obtained by combining multiple noisy evaluations
- We could evaluate systematically each candidate solution for a fixed number of times → not efficient from a computational perspective
- We want to dedicate more computational time to evaluate promising solutions and eliminate as quickly as possible the “lucky” ones → each candidate solution might have been evaluated a different number of times when compared
- In GA good and robust candidate solutions survive over generations; in PSO they survive in the individual memory
- Use dedicated functions for aggregating multiple evaluations: e.g., minimum and average or generalized aggregation functions (e.g., quasi-linear weighted means), perhaps combined with a statistical test
**GA**

1. Generate initial population randomly
2. Evaluate the initial population
3. Selecting parents according to a given selection scheme
4. Apply crossover to pairs of selected parents and generate offspring
5. Apply mutation gene wise to each individual in the population and generate more offspring
6. Evaluate the offspring
   - If the fitness is not deterministic, also re-evaluate the original population
7. Select the best individuals from the offspring and original population to generate the new population
8. Last generation?
   - Yes: End
   - No: Go back to step 2

**PSO**

1. Generate initial swarm and velocities randomly
2. Evaluate the initial swarm, using the initial performance as the initial personal best and using that to find the initial neighborhood best
3. Update velocities of particles using personal best and neighborhood best
4. Apply velocities to positions of particles
5. Evaluate new particle positions
   - If the fitness is not deterministic, also re-evaluate the personal best positions
6. Find the new personal best for each particle and use that to find the new neighborhood bests
7. Last iteration?
   - Yes: End
   - No: Go back to step 2
Testing Noise-Resistant GA and PSO on Benchmarks

• Benchmark 1: generalized Rosenbrock function
  – 30 real parameters
  – Minimize objective function
  – Expensive only because of noise

• Benchmark 2: obstacle avoidance on a robot
  – 22 real parameters
  – Maximize objective function
  – Expensive because of noise and evaluation time
Benchmark 1: Gaussian Additive Noise on Generalized Rosenbrock

\[ f'_j(\bar{x}) = \mathcal{N}(0, \sigma^2) + f_j(\bar{x}) \]

**Fair test:** same number of evaluations candidate solutions for all algorithms (i.e. n generations/iterations of standard versions compared with n/2 of the noise-resistant ones)
Benchmark 2: Obstacle Avoidance on a Mobile Robot

- **Similar** to [Floreano and Mondada 1996]
  - Discrete-time, single-layer, artificial recurrent neural network controller
  - Shaping of neural weights and biases (22 real parameters)
  - fitness function: rewards speed, straight movement, avoiding obstacles

- **Different** from [Floreano and Mondada 1996]
  - Environment: bounded open-space of 2x2 m instead of a maze

\[ F = V \cdot (1 - \sqrt{\Delta v}) \cdot (1 - i) \]
\[ 0 \leq V \leq 1, \ 0 \leq \Delta v \leq 1, \ 0 \leq i \leq 1 \]

V = average wheel speed, \( \Delta v \) = difference between wheel speeds, i = value of most active proximity sensor

Actual results: [Pugh J., EPFL PhD Thesis No. 4256, 2008]
Similar results: [Pugh et al, IEEE SIS 2005]
Baseline Experiment: Extended-Time Adaptation

- Compare the basic algorithms with their corresponding noise-resistant version
- Population size 100, 100 iterations, evaluation span 300 seconds (150 s for noise-resistant algorithms) → 34.7 days
- Fair test: same total evaluation time for all the algorithms
- Realistic simulation (Webots)
- Best evolved solutions averaged over 30 evolutionary runs
- Best candidate solution in the final pool selected based on 5 runs of 30 s each; performance tested over 40 runs of 30s each
- Similar performance for all algorithms

[Pugh J., EPFL PhD Thesis No. 4256, 2008]
Where can noise-resistant algorithms make the difference?

- Limited adaptation time
- Hybrid adaptation (simulation/hardware in the loop)
- Large amount of noise

Notes:
- all examples from shaping obstacle avoidance behavior
- best evolved solution averaged over multiple runs
- fair tests: same total amount of evaluation time for all the different algorithms (standard and noise-resistant)
Limited-Time Adaptation Trade-Offs

- Total adaptation time = 8.3 hours (1/100 of previous learning time)
- Trade-offs: population size, number of iterations, evaluation span
- Realistic simulation (Webots)

Varying population size vs. number of iterations

Especially good with small populations

[Pugh J., EPFL PhD Thesis No. 4256, 2008]
Hybrid Adaptation with Real Robots

• Move from realistic simulation (Webots) to real robots after 90% learning (even faster evolution)
• Compromise between time and accuracy
• Noise-resistance helps manage transition

[Pugh J., EPFL PhD Thesis No. 4256, 2008]
Increasing Noise Level – Set-Up

• **Scenario 1**: One robot learning obstacle avoidance
• **Scenario 2**: One robot learning obstacle avoidance, one robot running pre-evolved obstacle avoidance
• **Scenario 3**: Two robots co-learning obstacle avoidance

Idea: more robots more noise (as perceived from an individual robot); no “standard” com between the robots but in scenario 3 information sharing through the population manager!

[1x1 m arena, PSO, 50 iterations, scenario 3]

[Pugh et al, IEEE SIS 2005]
Increasing Noise Level – Sample Results

[Pugh et al, IEEE SIS 2005]
Not only Obstacle Avoidance: Evolving More Complex Behaviors
Evolving Homing Behavior
(Floreano and Mondada 1996)

Set-up

Robot’s sensors
Evolving Homing Behavior

- **Fitness function:**

\[ \Phi = V (1 - i) \]

- \( V = \) mean speed of wheels, \( 0 \leq V \leq 1 \)
- \( i = \) activation value of the sensor with the highest activity, \( 0 \leq i \leq 1 \)

- Fitness accumulated during life span, normalized over maximal number (150) of control loops (actions).
- No explicit expression of battery level/duration in the fitness function (implicit).
- Chromosome length: 102 parameters (real-to-real encoding).
- Generations: 240, 10 days embedded evolution on Khepera.
Evolving Homing Behavior

Fitness evolution

Evolution of # control loops per evaluation span

Battery recharging vs. motion patterns

- Battery energy
- Left wheel activation
- Right wheel activation

Reach the nest -> battery recharging -> turn on spot -> out of the nest
Evolved Homing Behavior
Evolving Homing Behavior

Activation of the fourth neuron in the hidden layer

Firing is a function of:

- battery level
- orientation (in comparison to light source)
- position in the arena (distance from light source)
Not only Control Shaping: Automatic Hardware-Software Co-Design and Optimization in Simulation and Validation with Real Robots
Moving Beyond Controller-Only Evolution

• Evidence: Nature evolve HW and SW at the same time …

• Faithful realistic simulators enable to explore design solution which encompasses co-evolution (co-design) of control and morphological characteristics (body shape, number of sensors, placement of sensors, etc.)

• GA (PSO?) are powerful enough for this job and the methodology remain the same; only encoding changes
Evolving Control and Robot Morphology  
(Lipson and Pollack, 2000)

http://www.mae.cornell.edu/ccsl/research/golem/index.html

- Arbitrary recurrent ANN
- Passive and active (linear actuators) links
- **Fitness function**: net distance traveled by the centre of mass in a fixed duration

Example of evolutionary sequence:
Examples of Evolved Machines

**Problem:** simulator not enough realistic (performance higher in simulation because of not good enough simulated friction; e.g., for the arrow configuration 59.6 cm vs. 22.5 cm)
Not only Control Shaping: Automatic Hardware Design and Optimization in Simulated Traffic Systems
Evolving Sensor Configuration

• Number $n$
• Type
  – Range $\rho$
  – Cone of view $\delta$
  – Cost $f(\rho, \delta)$
• Placement
  – Position $\phi$
  – Orientation $\theta$
Multi-Vehicle Simulation Tools

2001-2003: Webots 2, kinematic simulator; differential wheel vehicles; more later in the course
Vehicle Occurrence Probability

- 1-D
- 2-Dimensional

[Graph showing vehicle occurrence probability with axes labeled for angle in degrees and x, y in meters.]
Evaluation Tests – Deterministic

Static 1D Full Coverage 2D Full Coverage

Vehicles to be detected

Test vehicle Detection region
Evaluation Tests – Quasi-static

1D Quasi-static

2D Quasi-static

Test vehicle

Detection region

Vehicles to be detected
## Relative Time Cost

<table>
<thead>
<tr>
<th>Evaluation Tests</th>
<th>Time Cost</th>
<th>Relative Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Static</td>
<td>4.6 s</td>
<td>1</td>
</tr>
<tr>
<td>1D Full Coverage</td>
<td>9.7 s</td>
<td>2.1</td>
</tr>
<tr>
<td>2D Full Coverage</td>
<td>10.3 min</td>
<td>134</td>
</tr>
<tr>
<td>1D Quasi-static</td>
<td>15.6 s</td>
<td>3.4</td>
</tr>
<tr>
<td>2D Quasi-static</td>
<td>8.9 min</td>
<td>116</td>
</tr>
<tr>
<td>Embodied (kinematic)</td>
<td>10.4 hours</td>
<td>8100</td>
</tr>
</tbody>
</table>

Example: 2500 evaluations conducted on computers with 1.5 GHz AMD CPU
Fitness Function

$$Fitness(\omega, s) = \left( \frac{\mu_{cost}^s + \omega \cdot \mu_{coverage}^s}{1 + \omega} \right)^{\frac{1}{s}}$$

$$\omega = \frac{\omega_{coverage}}{\omega_{cost}}$$

Coverage = \sum_{i=1}^{V} k_i \text{PDF}(\alpha_i, r_i)$$

Cost = \sum_{i=1}^{n} cost_i$$

cost_i = f(\rho_i, \delta_i)$$

\mu_{xx}: fuzzy preference on factor xx
(0: totally unacceptable; 1: completely acceptable)

\omega_{xx}: weight on factor xx

s: degree of compensation

V: actual number of vehicles been in the detection region during an evaluation span

k_i: 1 if the ith car at distance r_i and approaching angle \alpha_i detected; 0 if not detected

n: number of sensors used

\rho_i: ith sensor’s range

\delta_i: ith sensor’s cone of view
Preference Functions

- Preference function given by the designer and/or the customer on all relevant criteria of a candidate solution
- Expressed here with fuzzy sets (0: totally unacceptable; 1: completely acceptable)
Sample Results

- Fitness evolution process and the final best design

\[ s = 0, \ \omega = \frac{3}{17} \]

Coverage = 53%, Total_cost = 4.6
Sample Results

- Fitness evolution process and the final best design

\[ s = -\infty, \omega \text{ arbitrary} \]

Coverage = 82\%, Total\_cost = 7.7
Sample Results

- Fitness evolution process and the final best design

\[ s = 0, \ \omega = 4 \]

Coverage = 98%, Total\_cost = 11.9
Sample Results

• Evolved approximate Pareto frontier for the design trade-offs present in the case study
• Each data point represents the best result of one particular evolutionary experiment under a given combination of \( \omega \) and \( s \)

[Zhang et al., Research in Engineering Design, 2008]
Conclusion
Take Home Messages

• Machine-learning algorithms can be classified as supervised, unsupervised, and reinforcement-based (evaluative)
• Evaluative techniques are key for robotic learning
• Two robust multi-agent metaheuristics are GA and PSO
• PSO is a younger technique than GA but extremely promising
• Computationally efficient, noise-resistant algorithms can be obtained with a simple aggregation criterion in the main algorithmic loop
• Machine-learning techniques can be used not only for design and optimization of controllers but also of further hardware features with the help of appropriate simulation tools
Books


Papers