Multi-Robot Coordination

Chapter 11
Objectives

- To understand some of the **problems being studied** with multiple robots
- To understand the **challenges involved** with coordinating robots
- To investigate a simple **behaviour-based self-organization** strategy for a common application
- To investigate a **simple communication** strategy
What’s in Here?

- Multi-Robot Coordination: Purpose and Issues
  - Advantages and Disadvantages of Multiple Robots
  - Types of Research and Disciplines
  - Role of Learning

- The Foraging Problem
  - What is it?
  - Explicit Distribution
  - Implicit Distribution
  - Improvement in Distribution

- Hierarchical Communication
  - What is it?
  - Various Schemes
    - Random
    - Sequential
    - Vector
    - Focused Averaging
Multi-Robot Coordination: Purpose and Issues
Multiple Robots

There are **advantages** when using multiple robots:

- **larger range** of task domains
- **greater efficiency**
- **improved system performance**
- **fault tolerance**
- **lower economic cost**
- **ease of development**
- **distributed** sensing and action
Multiple Robots

There are also disadvantages / challenges:

- performance depends on issues involving interaction between robots
- interactions complicate development
- difficult to model group behaviors from top down (i.e., centralized control) when environment is unknown and/or dynamic
- sensor and/or physical interference
- need lots of batteries!
5 major themes of robot group research:

- **Group control architecture**
  - decentralization and differentiation

- **Resource conflict resolution**
  - e.g., space sharing

- **Origin of cooperation**
  - i.e., genetically-determined social behavior or interaction-based cooperative behavior

- **Learning**
  - e.g., control parameter tuning for desired cooperation

- **Geometric problem solving**
  - e.g., geometric pattern formation

A typical research paper will focus on **only one theme** (or aspect) of group robotics.
Research

What kinds of problems have been studied:

- Multi-robot path planning
- Traffic control
- Formation generation, keeping and control
- Target tracking
- Multi-robot docking
- Box-pushing
- Foraging
- Multi-robot soccer
- Exploration and localization
- Transport
Disciplines

There are three disciplines that are most critical to the development of robotic agents:

- **Distributed Artificial Intelligence**
  - distributed Problem Solving or Multi-Agent Systems
  - considers how tasks can be divided among robots which share knowledge about problem and evolving solutions.

- **Distributed Systems**
  - focus on distributed control addressing deadlock, message-passing, resource allocation etc…

- **Biology**
  - bottom-up approach where robots follow simple reactive rules
  - Interaction between robots results in complex emergent behavior
Learning and Adapting

- Robots perform for a certain period of time without human supervision in order to solve a problem.
  - Must be able to deal with dynamic changes in environment and their own performance capabilities.

- Learning, evolution, and adaptation allow a robot to improve its likelihood of survival and its task performance in the environment:
  - **adaptation** – how a robot learns by making adjustments.
  - **learning** – helps one robot adapt to the environment.
  - **evolution** – helps many robots adapt to the environment.
Evolution vs. Learning

- **Evolution**: process of selective reproduction and substitution based on the existence of a distributed population of vehicles
  - does not perform well when certain environmental changes occur that are different from evolved solutions

- **Learning**: a set of modifications taking place within each individual during its own lifetime
  - often takes place during an initial phase when task performance is considered less important
  - control policy used that gives reasonable performance robot “team” gradually improves over time.
Overview Summary

- There are many aspects of multi-robot coordination.
- Robots that perform well together in one kind of environment will perform poorly in others.
- To be useful, multi-robot strategies must:
  - be “designed” and “fine-tuned” for particular applications
  - explicitly / implicitly distribute the work among the robots
  - consider both sensory and environmental interference from other robots
  - be able to operate under unexpected situations
  - be cost-effective
Multi-Robot coordination strategies is a huge topic
  - too much to cover in this course

We will consider:
  - self-organization for simple foraging applications
  - hierarchical communication to focus coverage

We will look at simulated results:
  - robots will be reactive and use instinctive behaviors
  - analyze the performance over time
  - combine different types of robots
The Foraging Problem
Foraging

- Consider a common problem studied in robotic colonies, *foraging*:
  - gathering/collecting items
    - possibly bringing them to some specific location(s) (e.g., to particular room) or general locations(s) (e.g., to outer walls).
  - there are many variations of this problem

- We will consider a specific instance:
  - robots can detect when it finds an item and can push it to some location (or pick it up and drop it off).
  - robots will be encoded with a fixed, instinctive behavior and thus will not learn “how” to forage.
Foraging

Consider allowing robots to move randomly in an environment with no cooperation.

Robots must find forage items (e.g., when passing over them) and bring them to the boundaries.

- Robots may collide, which may interrupt the forage procedure of a robot.
- Eventually, over time, each forage item will be found by some robot:
Foraging

- As more robots are used, the speed of forage completion increases.

- The performance decreases when the forage items are not evenly distributed.
  - this is because robots are not directed towards forage items, only finding them by chance.
Foraging

- Intuitively, performance can be improved by:
  - reducing collisions (or interference) between robots
  - preventing robots from traveling over the same areas
  - directing robots towards clusters of forage items

- The obvious way of reducing collisions and preventing duplicate travel is to distribute robots by explicitly assigning each one a particular area in the environment in which to forage.
  - environment broken down into “equal-sized” areas which are assigned to individual robots
Foraging – Explicit Distribution

- This strategy has **advantages:**
  - ensure *even distribution* of robots
    - good when items to be foraged are evenly distributed randomly
  - minimizes *sensor interference* and physical collisions between robots

- **disadvantages:**
  - requires robots to “know” and *maintain specific positions*
    - requires knowledge of environment
    - expensive sensors ?? (e.g., GPS)
    - expensive computation (e.g., position estimation)
  - can be *inefficient* if forage items are clustered
Foraging – Explicit Distribution

- A simple way of determining the foraging areas for each robot is to base the regions on the dual graph:

  Recursively divide dual graph in “half” until number of regions matches the number of robots:

Each robot remains in its own designated area.
There are multiple ways to split the dual graph by finding an edge that evenly splits:

- **links** – # of dual graph links
  - simple and fast, assuming a nice triangulation

- **area** – area covered by dual graph triangles
  - best if robots need to perform coverage algorithms or searching with uniform distribution of foraging items.

- **perimeter** – perimeters of dual graph triangles
  - good if robots are to patrol outer boundaries of their environment
Performance (i.e., speed of forage completion) is highly dependant on shape of environment and location of forage items. 

With clustered forage items, most robots become useless if forced to remain in a particular area.

With forage items evenly distributed, robots work effectively in near optimal configuration, provided that robots do not have to leave their environment to complete the task.
Foraging – Implicit Distribution

- Clearly, fixing the locations of each robot may not be the best choice if:
  - the distribution of forage items is not known to be random and evenly distributed
  - the robots must travel outside their areas to complete the forage task (i.e., to deliver their payload).

- A compromise is to hard-code specific behavioral rules into the robots that minimize their collisions and attempt to keep them distributed.
Foraging – Implicit Distribution

Consider robots with omni-directional beacons which are detectable from other nearby robots:

- robots avoid moving towards nearby beacons
- intuitively, robots should remain separated/distributed

When other robot detected within sensor range, robot moves in opposite direction.

With multiple beacons, either move away in combined vector direction or away from strongest signal.

Although robots may still re-encounter other robots during their movements, in general they remain distributed.
A comparison of these schemes shows that:

- for evenly spread forage items there is no significant advantage of either scheme in terms of forage completion time and the simple random movement seems to do well.

- for clustered forage items the fixed area scheme performs poorly with few robots and the repel scheme performs better.
Foraging – Improvement

A more significant improvement can be made if something is known about the forage items (e.g., they are clustered).

- can “signal” other robots when item is encountered
- leave signal on until:
  - fixed amount of time elapses
  - other robots come nearby
- can either wait stationary or continue moving
Foraging – Improvement

Consider five “beacon attraction” schemes:

- **Always On**
  - beacon is always on, robot keeps moving

- **Timed Out Stationary**
  - beacon on for fixed time, robot waits stationary until beacon timeout

- **Timed Out Moving**
  - beacon on for fixed time, robot keeps moving

- **Until Near**
  - beacon on until robot nearby, robot waits stationary until another robot comes nearby

- **Until Near or Timed Out**
  - beacon on for fixed time, robot waits stationary until beacon timeout or until another robot comes nearby

Robots may get into a deadlock situation.
Here is the basic idea behind the attraction code:

\[
\text{REPEAT} \{
    \text{int desiredDirection = direction of closest/strongest beacon signal;}
    \text{IF (desiredDirection != null)} \{
        \text{boolean collisionDetected = read front collision sensors;}
        \text{IF (collisionDetected)} \{
            \text{Turn away from obstacle;}
        \}
        \text{Turn towards desiredDirection}
    \}
    \text{ELSE} \{
        \text{wander (i.e., move forward or turn randomly)}
    \}
\}
\]

Depending on sensor. The desired direction may be that of the strongest signal (if many beacons sensors are mounted in a circular fashion), or may be a direction representing a combination of multiple signals. Usually, the direction will be one of 8 to 16 fixed directions around the robot.

\[
\text{IF (a forage item is found)} \{
    \text{Turn on my beacon;}
    \text{Wait for XXX seconds;}
    \text{Turn off my beacon;}
\}
\]

Add this code for the \texttt{TimedOutStationary} scheme

\[
\text{IF (a forage item is found)} \{
    \text{Turn on beacon;}
    \text{counter = 5000; //msec}
\}
\]

Add this code instead for the \texttt{TimedOutMoving} scheme

\[
\text{IF (--counter == 0)} \{
    \text{Turn off beacon;}
\}
\]
What about performance?

Movement while beacon is on, is best strategy. Up to 3x faster than random movement here.

Attraction with timed beacon always improves performance.

Attracting all the time, can be worse than moving randomly.

Increasing Time ===>
Foraging – Improvement

- Even when varying the number of robots, the attraction scheme performs well:

![Graphs showing scheme comparison for 4 and 100 robots.]

The time scales between the graphs is different, in order to accentuate the differences in the schemes.
Of course, in non-clustered environments, the attraction scheme performance degrades and actually reduces efficiency over random scheme:

Recall that repel scheme works best in unclustered environments.

Attraction schemes perform worse than simple random movement.
What about environments with both clustered items AND spread out items?

Performance is near to random … but provides only a small improvement.
Foraging – Improvement

- Can mix various kinds of robots:
  - e.g., some attract, some repel

Certainly, less robots are attracted to cluster, so in clustered environment, there is a performance tradeoff when combining robot types.

Combining 6 random with 6 attract robots performs best despite type of environment!!
Other Similar Problems

- Similar attraction/repel strategies can be implemented for other problem scenarios such as coordinated mapping, searching, patrolling, floor cleaning etc.
  - same principles apply, but results may differ.

- As seen, using heterogeneous groups (i.e., mixing different kinds of robots) may prove to be the most robust and efficient solution overall.

- Experimentation helps to tweak solutions:
  - wanna do an honours project or a Master’s thesis?
Hierarchical Communication
Another important issue with respect to multi-robot algorithms has to deal with communications:

- do the robots need to communicate (e.g., send data) ?
- is there any advantage to doing so ?
- how often should they communicate ?
- should there be unlimited communication between robots or should there be restrictions (i.e., groups) ?

We will look here at one aspect of using hierarchical communication.
Hierarchical Communication

Consider robots organized into a hierarchy:

- Each robot belongs to a group and all group members can communicate to a group “leader” via wireless communication.
- The leaders are also grouped together with a higher level leader to which they communicate.
Hierarchical Communication

• Within a hierarchy, worker robots must always remain within communication range:
  - allows data to be transmitted to leader (e.g., map data)
  - allows leader to send commands at any time (e.g., new directions and updated task assignments)
  - allows quick docking for battery recharging, working in shifts etc…
  - an warning buffer zone should be used to inform worker to turn back.
Hierarchical Communication

- A main issue with bottom-up behavior-based programming is that only *local information* (i.e., information from a robot’s own sensors) is usually available.

- With such a hierarchical scheme, lower level robots can be given global knowledge of the environment and/or of task completion.
  - should provide benefit over no-communication schemes for more complex problems
  - can allow “steering” of robots to accomplish task more efficiently.
Hierarchical Schemes

- Consider robots moving randomly to cover a simple environment:
  - good enough to investigate the general problem of robot coverage under various communication schemes.
  - more efficient schemes can be used to cover environment and techniques can be “tweaked” to each application.
  - random coverage actually performs well over time.

Random coverage of 4 robots over time.
Hierarchical Schemes

- Now consider a leader with 4 worker robots:
  - worker robots move randomly within leader’s communication range:
  - we can restrict worker movements to fixed or variable-sized wedges/quadrants:

Robots all move randomly within communication range.

Robots may cross over into other quadrants, but treat it as out of range.
Hierarchical Schemes

- Leader must also move in order to cover whole environment properly.

- Consider various leader movement schemes:
  - Random: move in random direction
  - Sequential: move along a fixed path in sequence
  - Vector: move in direction towards quadrant that had most “out of safe zone” occurrences
  - Toward Average: vector scheme with added “pull” towards leader’s average location
  - Away From Average: vector scheme with added “push” away from leader’s average location
Hierarchical Scheme - Sequential

- The basic sequential scheme works as follows:
  - Leader moves slower than workers (e.g., 1/10\textsuperscript{th} of speed)
  - Leader heads towards next location in some sequence (e.g., along a predetermined path)
  - Leader may remain at each location for a while or leave immediately.
  - Timeout may be used if location is not reached within certain time limit.

Leader moves along path, while workers move randomly within the “safe” range.

Necessary in order to avoid getting stuck behind obstacles.

Good if need to unload workers, then reload and transport to new site.

Necessary in order to avoid getting stuck behind obstacles.
Hierarchical Scheme - Vector

The basic vector scheme works as follows:

- Leader moves slower than workers (e.g., 1/10th of speed)
- Each time worker leaves “safe” range, a counter is incremented
- Leader computes 4 vectors facing 4 quadrants with magnitudes equal to these counters
- Leader moves
  - in combined direction of these vectors, or
  - in direction of strongest magnitude vector

Leader moves in combined vector direction.
Hierarchical Scheme – Average Vector

The average vector scheme works as follows:

- **Same 4 vectors** as Vector scheme are used
- Leader also keeps track of its overall average position
- Leader computes 1 new vector facing either towards or away from the global average according to its current location
- Leader includes this new vector in its computations
- Magnitude of global average vector set to scalar multiple of maximum of other vectors (e.g., 2x, 1x, ½x, etc…)

Average vector, set to 1x maximum
Hierarchical Results

Results from the **Random** movement scheme:

- Leader moves randomly, while workers stay nearby. This strategy may not reach all parts of the environment.
- Too much clustering on edges.
Hierarchical Results

- Results from the 4-Point **Sequential** scheme:

  Leader moves to predefined locations (in this case, 4 “corners”), while workers stay nearby.

  Nice coverage in general, but corners are missed. How can we fix this?
Hierarchical Results

Results from the **Vector** scheme:

- Leader moves toward the direction of the worker that was out of safe range the most times.
- Performs ok, but not better than without communication.
Hierarchical Results

Results from the **Toward Average Vector** scheme:

- good for applications such as focused searching in which the likelihood of success is localized about some known location.

- Can really focus attention of workers around a specific area.
- Can keep less focus to allow outward expansion.
- Can form search "rings" by varying magnitude over time.

2x Attraction Magnitude 1x Attraction Magnitude ½ x Attraction Magnitude
Hierarchical Results

Results from *Away From Average Vector* scheme:

- good for applications such as mapping to “force” exploration away from previously mapped areas.

Can use a “hint” of focus to allow more randomness. Can keep less focus to allow inward expansion. Can really focus attention of workers away from a specific area.
Hierarchical Results

Results in environments with obstacles:

- Good overall coverage, but does not consider obstacles as different.
- Can provide a better coverage around obstacles resulting in more accurate mapping.
- Can provide a coverage more focused along path (in this case around outer obstacle cluster).

No Communication  Vector Scheme  Sequential Scheme
Summary

You should now understand:

- The issues involved with coordinating multiple robots
- How to produce self-organization using simple behaviors
- The simple foraging problem and how to improve performance in various ways
- How to provide simple hierarchical communication to focus multi-robot coverage.