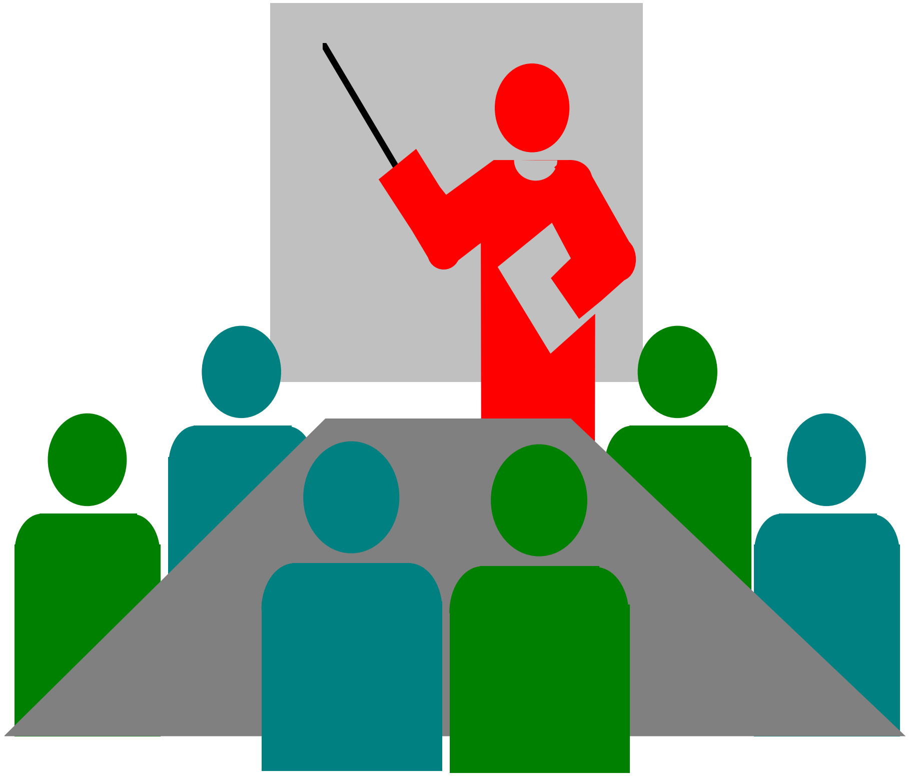
Topic 9: Building Robot Behaviours



9.1 Introduction

Behaviour-based Robotics:

- ❑ The world is fundamentally unknown and dynamically changing.
- ❑ It is wasting time to over plan a sequence of actions
- ❑ Key idea: to develop a library of useful behaviours (=controllers).
- ❑ To switch among controllers or behaviours in response to changes in environments

Behaviour definition:

Robot behaviour is a control action based on a stimulus from a dynamic and unknown environment.

9.1 Introduction

Questions:

- What are behaviours for a robot?
- Where do robot behaviours come from?
- How are these behaviours effectively co-ordinated?
- How are behaviours grounded to sensors and actuators?

A navigation example:

Consider a robot going to the kitchen. It may involve:

- finding the destination from its current position
- walking towards the destination without bumping into anything
- observing and giving way to someone who is on the way.
- coping with changes on the way & doing whatever is necessary
- reaching the destination timely and successfully

9.1 Introduction

9.2 Behaviour Expression

To implement the navigation task described above, we have to build a robot which has particular behaviours needed.

Expression of robot behaviours:

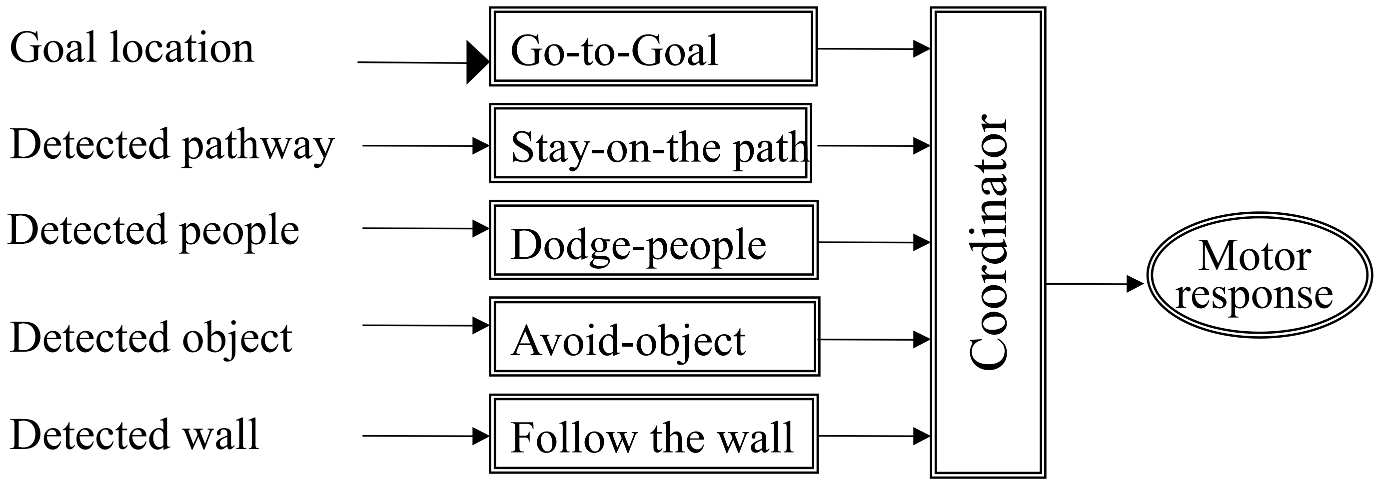
Several methods are available for expressing robotic behaviours, including

- *Stimulus-response (SR) diagrams*:
  -- used for graphic representations of specific behavioural configurations
- *Functional notation*:
  -- used for clarity in design of the systems
- *Finite state acceptor (FSA) diagrams*:
  -- used for temporal sequencing of robot behaviours
- *Formal method:*
  -- used as a common language for expressing robot behaviours

9.2.1 Stimulus-response (SR) diagrams

- SR is the most intuitive and the less formal method of expression.

For example:



- Motor response is corresponding to the behaviour with the highest priority.
- The coordinator is to coordinate different behaviours according to their priority.

9.2.2 Function notation

It is a mathematical method to express robot behaviours. For example:

$b(s) = r$ which means that behaviour ***b*** at a given stimulus ***s*** yields a response ***r***.

The kitchen navigation task can be expressed:

```
Coordinate-behaviours[
        go-to-kitchen(detect-kitchen-location),
        avoid-objects(detect-objects),
        dodge-people(detect-people),
        stay-on-path(detect-path),
        follow-the-wall(detect-wall)
] = motor-response
```

Coordinated functions permits a recursive formulation of robot behaviours.
It is easy to write in LISP, C/C++.

```
coordinate-behaviours[
        coordinate-behaviours(behavioural-set-1)
        ……
        coordinate-behaviours(behavioural-set-N)
]
```

9.2.3 Finite state acceptor (FSA) diagrams:

An FSA, *M*, is specified by: $M = (Q, \delta, q_0, F)$

where:

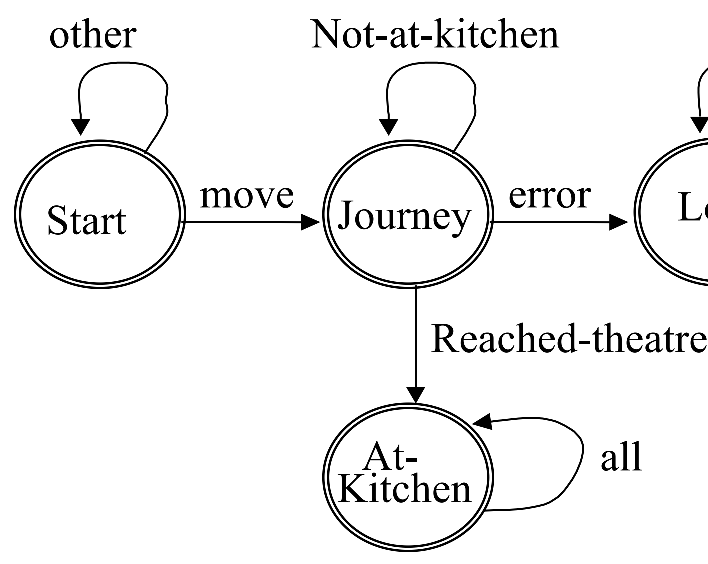
Q -- represents the set of allowable behavioural states.

δ -- is a transition function mapping the input & the current state to another, or even the same, state.

q_0 -- denotes the starting behavioural configuration.

F -- represents a set of accepting states, indicating completion of the sensorimotor task.

FSA used for the kitchen navigation example:



M = {{start, journey, lost, at-kitchen}, δ , start, {lost, at-kitchen}}

9.2.4 Formal Methods

Two typical formal methods:

The robot schema (RS) model

-- developed by Lyons and Arbib (1989)

The situated automata model

-- developed by Kaelbling and Rosenschein (1991)

- They can be used to verify designer's intentions.
- They can be facilitate the automatic generation of robotic control systems.
- They provide a complete common language for expressing robot behaviours.
- They provide a framework for conducting formal analysis of a specific program's properties, adequacy, and/or completeness.
- They provide support for high-level programming language design.

9.2.4 Formal Methods

The Robot Schema (*RS*) model:

- ***RS*** model expresses a sensor-driven robot behaviour which encodes its response to any input messages.
- Schemas communicate with each other via pre-defined input-output ports using synchronous message passing techniques.
- Schemas are aggregated via a nesting mechanism termed an assemblage.

<u>The way to build:</u>

- A process algebra is used to composite a network of schemas (behaviours).
- Process composition operators include methods for conditional, sequential, parallel and iterative structures.
- Preconditions are established for coordination operators to ensure a smooth control during execution.

9.2.4 Formal Methods

The *RS* model for the kitchen navigation:

Service-Robot = (Start_up; (done? , Journey) : At-kitchen)
Journey = (move-to-kitchen, avoid-objects, dodge-people,
stay-on-path, follow-the-wall)

where the sequential operator is denoted ';'(*semicolon*), the concurrency operator is denoted as ','(*comma*), the conditional operator is ':'(*colon*).

The main features:

- The Service-robot consists of a robot that, beginning from an initial start-up state, transitions to the Journey state, and remains there until it is at the kitchen indicated by a concurrent monitor process.
- The Journey state consists of the concurrent execution of the behaviours specified during travel from one location to the next.

9.2.4 Formal Methods

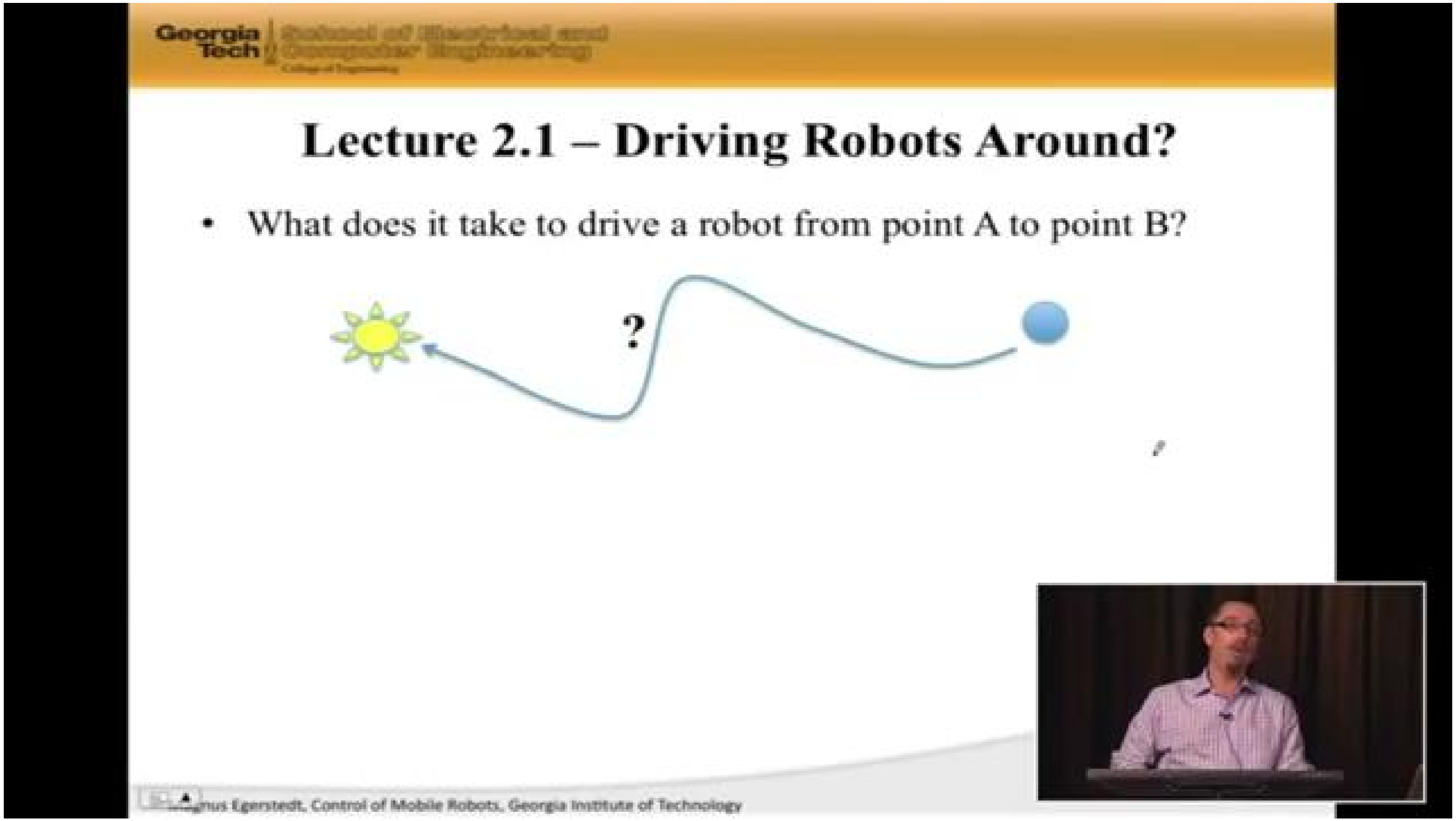
The *Situated Automata* (*SA*) model:

- The model employs logical formalisms corresponding to a robot's goals & intentions.
- The use of logic such as operators and, or, not and if, enables reasoning over the system leading to create high-level goals.
- A LISP-based system and the Gapps language recently developed are basic tools.

SA model for the kitchen navigation task:

```
(defgoalr (ach in-theatre)
          (if (not start-up)
              (maint (and  (maint move-to-kitchen)
                           (maint avoid-objects)
                           (maint dodge-people)
                           (maint stay-on-path)
                           (maint follow-the-wall)))
)
```

9.2.5 Behaviour Expression Video



9.3 Behaviour Encoding

A behaviour can be expressed as:

$$\beta : S \rightarrow R$$

where S -- the domain of all interpretable stimuli
R-- the range of possible responses
β-- mapping from the stimulus to possible responses

Three general categories of β:

- Null: The stimulus produces no motor response.
- Discrete: The stimulus produces a response from a set of prescribed choices (e.g. turn right, go-straight, stop, travel-at-speed-5).
- Continuous: The stimulus domain produces a motor response that is continuous over R's range (e.g. increase/decrease speed).

9.3 Behaviour Encoding

9.3.1 Discrete encoding

β is represented as a collection of *if-then* rules in the general form:

IF *antecedent* **Then** *consequent*

where the antecedent consists of a list of preconditions, and the consequent contains the motor responses (forward, backward, left…)

Examples:

If-then behaviour rules used for a wheelchair (Connell100)

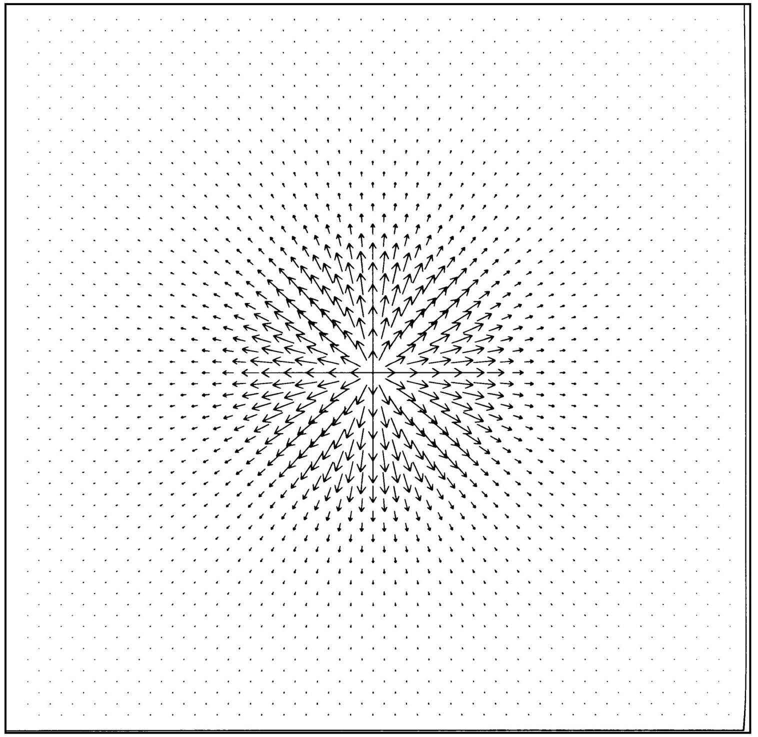
- Approach-- **If** an object is detected far away, **Then** go forward.
- Retreat -- **If** an object is nearby, **Then** move backward.
- Stymie -- **If** all the front IR sensors see an object, **Then** turn left.

9.3 Behaviour Encoding

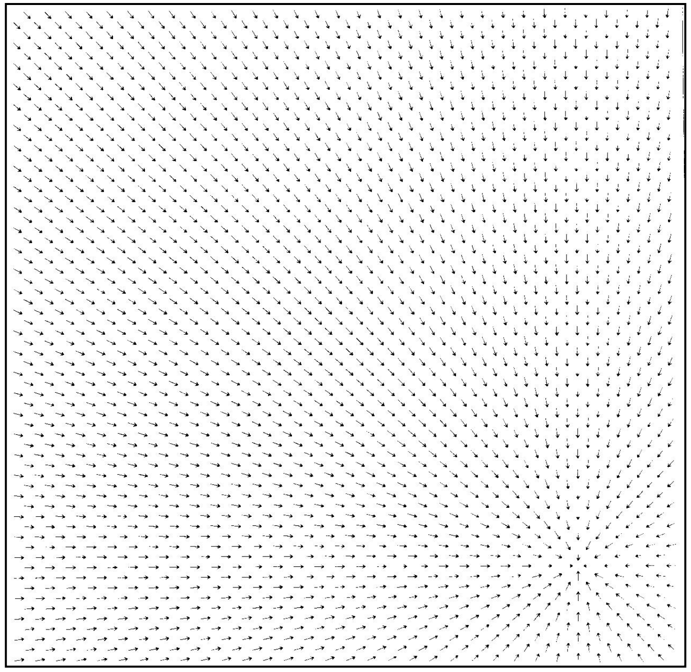
9.3.2 Continuous functional encoding

- This encoding technique is to transforms the sensory input into a continuous behaviour response.
- The potential fields method (Khatib & Krough).

$$Force \propto \frac{1}{Distance^2}$$



(a) A repulsive force field for an obstacle

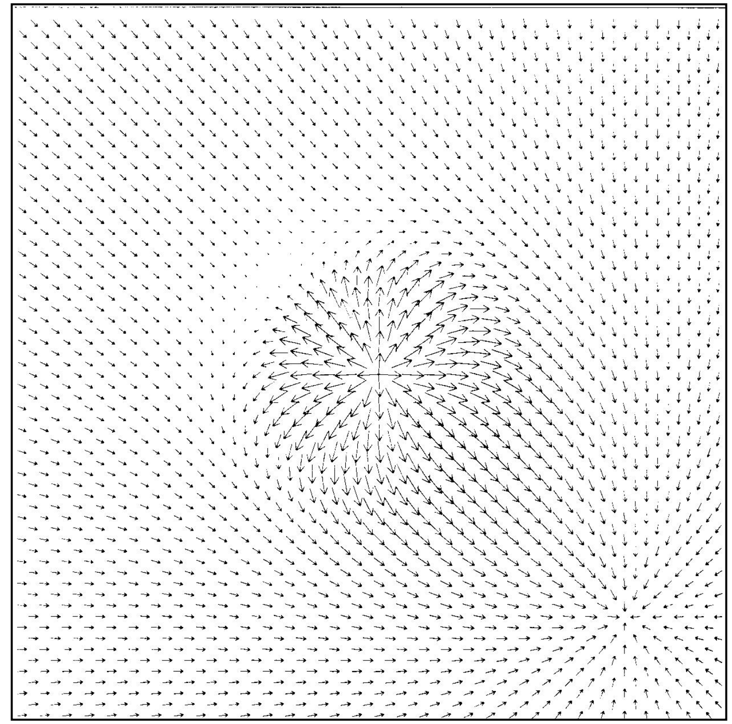


(b) A ballistic attraction field for a goal

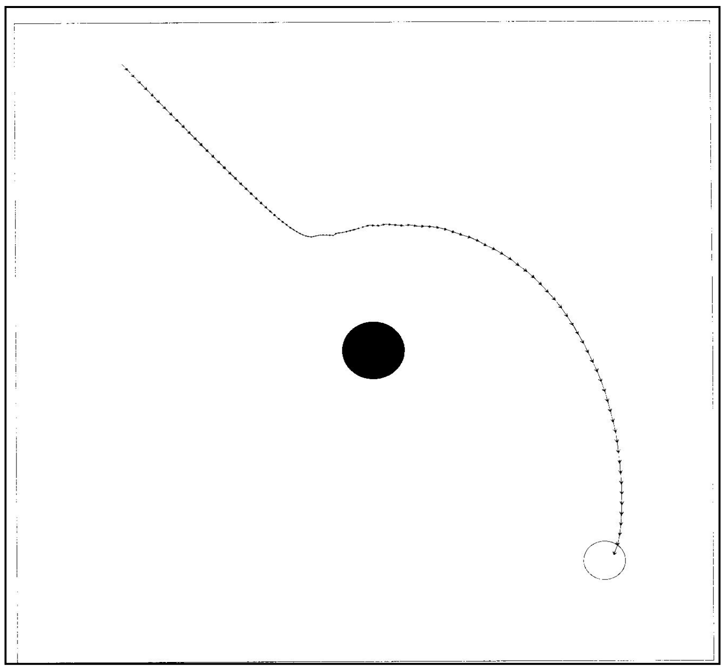
9.3 Behaviour Encoding

Main features:

- Potential fields encode a continuous navigation space through the sensed world, and provide an infinite set of possible actions.
- There is a local minimum problem in potential fields, where a robot may get stuck or oscillation.

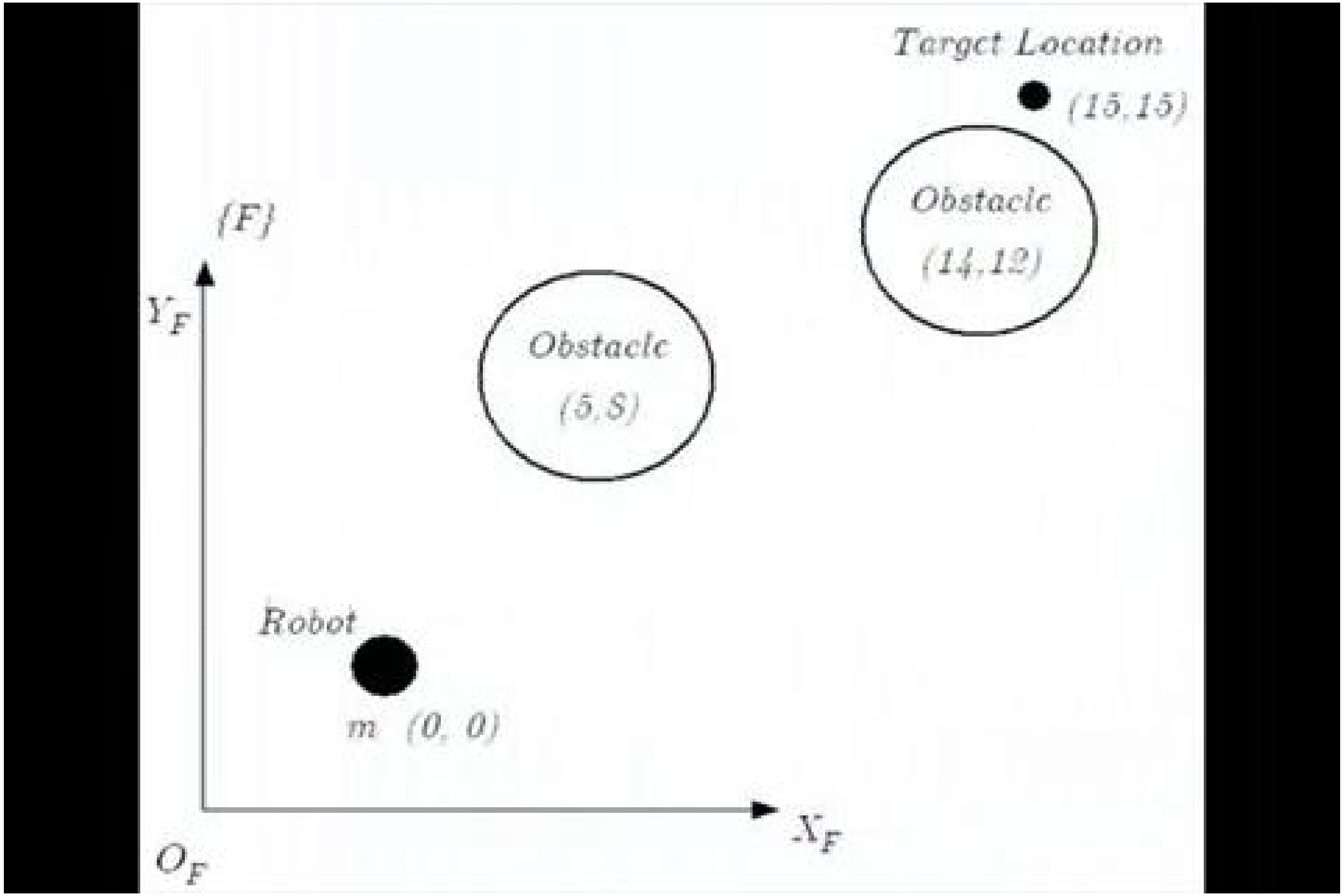


(c) A linear superposition of these two fields



(d) An example trajectory for a robot moving within this simple world

9.3 Behaviour Encoding



9.4 Behaviour Assembling

9.4.1 Behaviour Assembling

Since multiple behaviours may be concurrently active within a robotic system, we define additional notation as follows:

- **S** -- a vector of all stimuli $\mathbf{s}_i$ relevant for each behaviour β_i detectable at time *t*.
- **B** -- a vector of all active behaviours β_i at a given time *t*.
- **R --** a vector of all responses $\mathbf{r}_i$ generated by the set of active behaviours.
- **G** -- a vector encoding the relative strength or gain g_i of each active behaviour β_i.

$$\mathbf{R} = \begin{bmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \\ \mathbf{r}_3 \\ \vdots \\ \mathbf{r}_n \end{bmatrix}, \quad \mathbf{S} = \begin{bmatrix} \mathbf{s}_1 \\ \mathbf{s}_2 \\ \mathbf{s}_3 \\ \vdots \\ \mathbf{s}_n \end{bmatrix}, \quad \mathbf{B} = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \vdots \\ \beta_n \end{bmatrix}, \quad \mathbf{G} = \begin{bmatrix} g_1 \\ g_2 \\ g_3 \\ \vdots \\ g_n \end{bmatrix}$$

9.4 Behaviour Assembling

Definition: The overall robotic system response is defined as follows:

$$\rho = \mathbf{C}(\mathbf{G} * \mathbf{B}(\mathbf{S})) = \mathbf{C}(\mathbf{G} * \mathbf{R})$$

where **C** is a new behaviour coordination function.

Example: the kitchen navigation

Given the robot's current perceptions at time *t*, we have

$$\mathbf{S} = \begin{bmatrix} \mathbf{s}_1 \\ \mathbf{s}_2 \\ \mathbf{s}_3 \\ \mathbf{s}_4 \\ \mathbf{s}_5 \end{bmatrix} = \begin{bmatrix} (kitchen - location, 1.0) \\ (detected - object, 0.2) \\ (detected - people, 0.8) \\ (detected - path, 1.0) \\ (detected - wall, 0.0) \end{bmatrix}$$

9.4 Behaviour Assembling

The behaviour responses are represented as:

$$\mathbf{B}(\mathbf{S}) = \begin{bmatrix} \beta_{go-to-kitchen}(\mathbf{s}_1) \\ \beta_{avoid-object}(\mathbf{s}_2) \\ \beta_{dodge-student}(\mathbf{s}_3) \\ \beta_{stay-on-path}(\mathbf{s}_4) \\ \beta_{follow-the-wall}(\mathbf{s}_5) \end{bmatrix} \Rightarrow \begin{bmatrix} \mathbf{r}_{go-to-kitchen} \\ \mathbf{r}_{avoid-object} \\ \mathbf{r}_{dodge-people} \\ \mathbf{r}_{stay-on-path} \\ \mathbf{r}_{follow-the-wall} \end{bmatrix} = \mathbf{R}$$

We arbitrarily choose component vector magnitudes in **R** as follows

$$\mathbf{R}_{magnitude} = \begin{bmatrix} 1.0 & 0 & 0.8 & 1.0 & 0 \end{bmatrix}^T$$

where *avoid-object* and *defer-to-elder* are below threshold and generate no response. *go-to-theatre* and *stay-right/left* are at maximum strength.

9.4 Behaviour Assembling

Then we choose the gain vector as follows:

$$\mathbf{G} = \begin{bmatrix} g_{go-to-kitchen} \\ g_{avoid-object} \\ g_{dodge-people} \\ g_{stay-on-path} \\ g_{follow-the-wall} \end{bmatrix} = \begin{bmatrix} 0.8 \\ 1.2 \\ 1.5 \\ 0.4 \\ 0.8 \end{bmatrix}$$

Finally, we have

$$\rho = \mathbf{C}(\mathbf{G} * \mathbf{R}) = \mathbf{C}\left(\begin{bmatrix} g_1 * \mathbf{r}_1 \\ g_2 * \mathbf{r}_2 \\ g_3 * \mathbf{r}_3 \\ g_4 * \mathbf{r}_4 \\ g_5 * \mathbf{r}_5 \end{bmatrix}\right) \Rightarrow \mathbf{R}'_{magnitude} = \begin{bmatrix} 0.8 \\ 0 \\ 1.2 \\ 0.4 \\ 0 \end{bmatrix}$$

It should be noted that if using winner-take-all coordination strategy, the *dodge-people* behaviour will be chosen as it has a greatest magnitude.

9.4 Behaviour Assembling

9.4.2 Behaviour coordination

The coordination function **C** has two predominant classes:

competitive & cooperative

Competitive methods

- Competitive methods provide a means of coordinating behaviours response for conflict situation when two or more behaviours are active.
- Coordination is done through a *winner-take-all* strategy.
- There are mainly three popular competitive methods as follows:

  -- Priority-based coordination

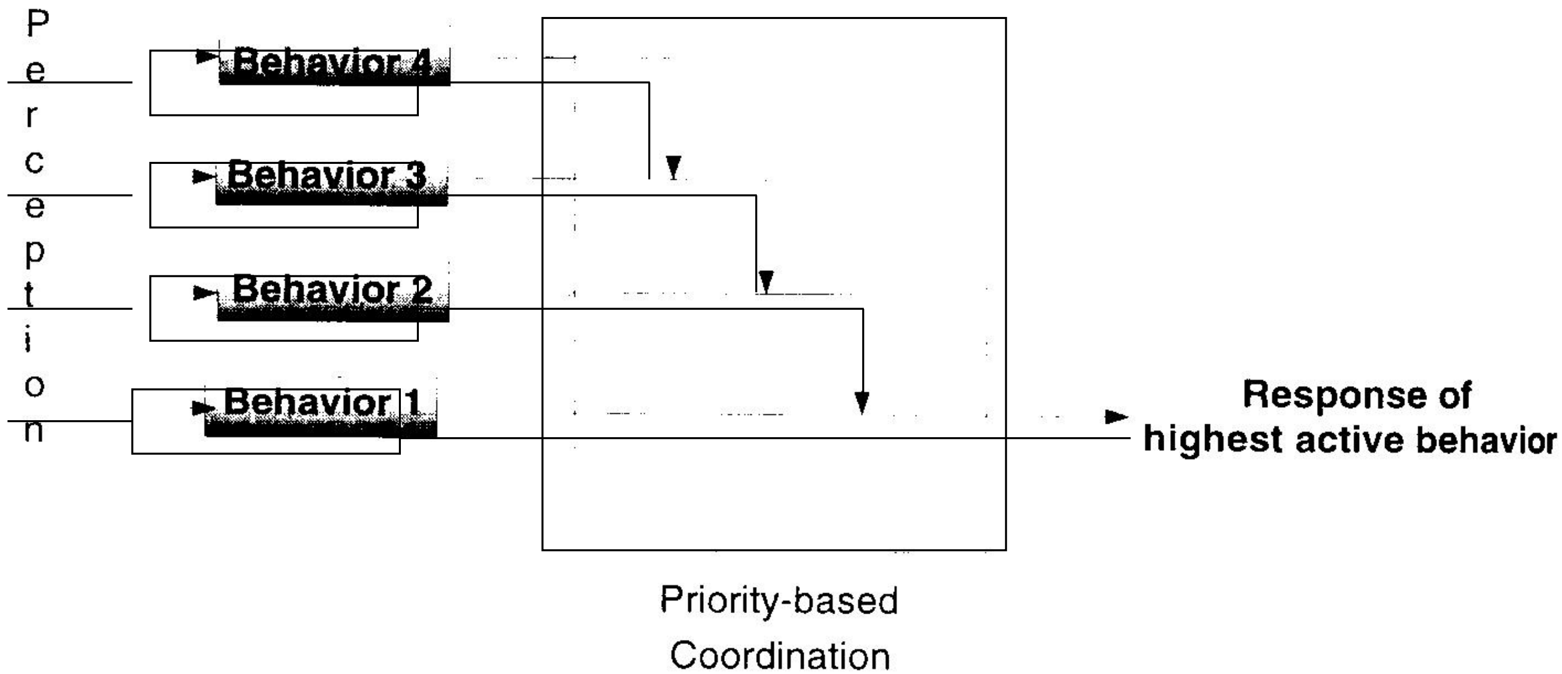
  -- Action-selection coordination

  -- Voting-based coordination

9.4 Behaviour Assembling

Priority-based coordination (*Brooks, 1986*)

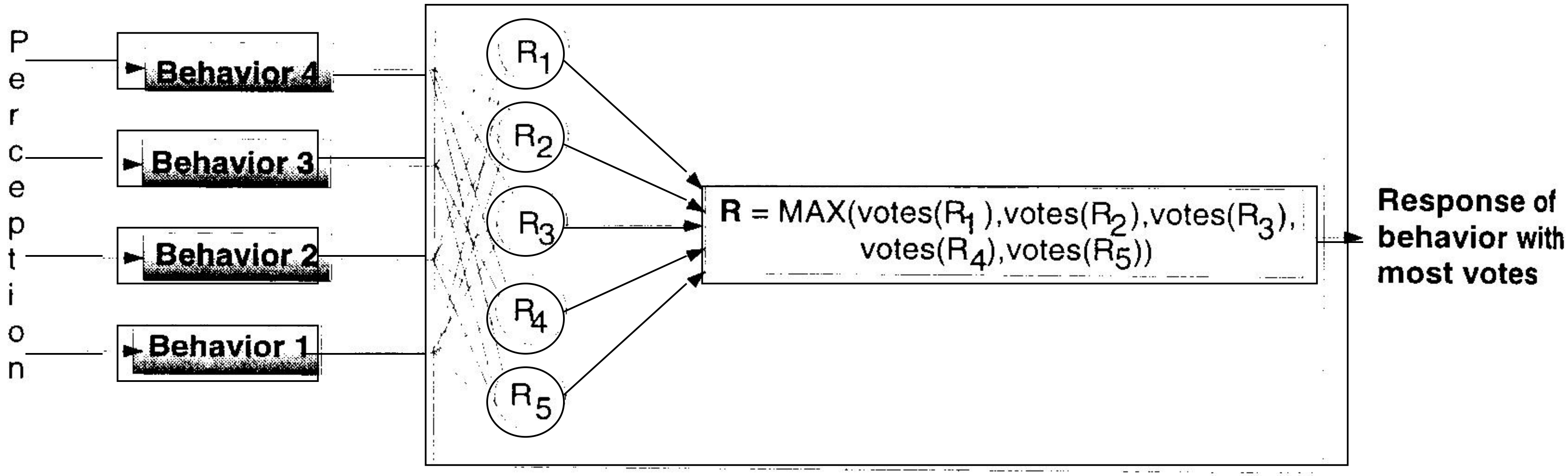
- A coordination function serves as an arbiter in a fixed priority.
- Only a single behavioural response is allowed at any time.



9.4 Behaviour Assembling

Voting-based coordination(*Rosenblatt & Payton*)

- A democratic competitive: voting for action
- The single response with the most votes is enacted.

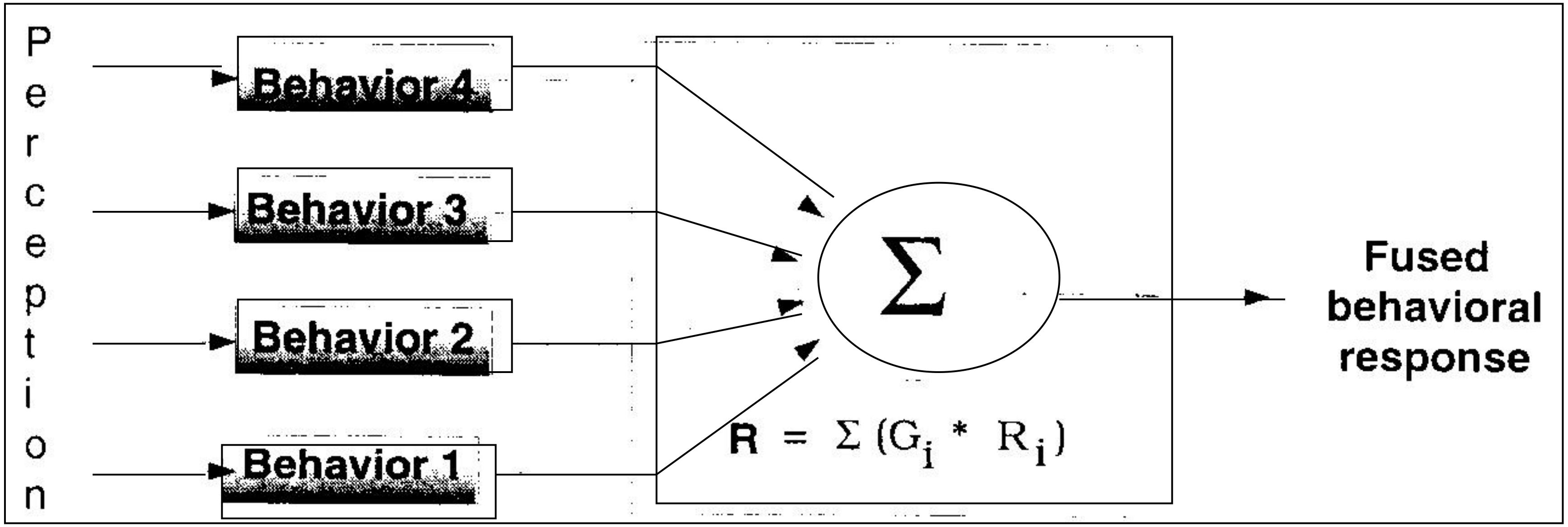


Voting-based Coordination

9.4 Behaviour Assembling

Cooperative methods

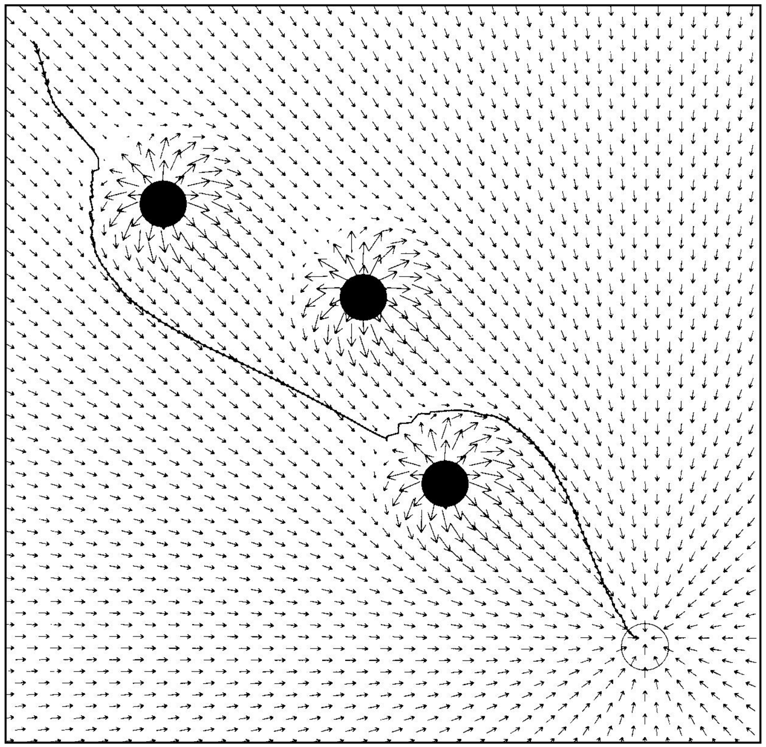
- Cooperative methods combines the outputs of multi behaviours.
- Vector addition & superpositioning are popular ways for combining.



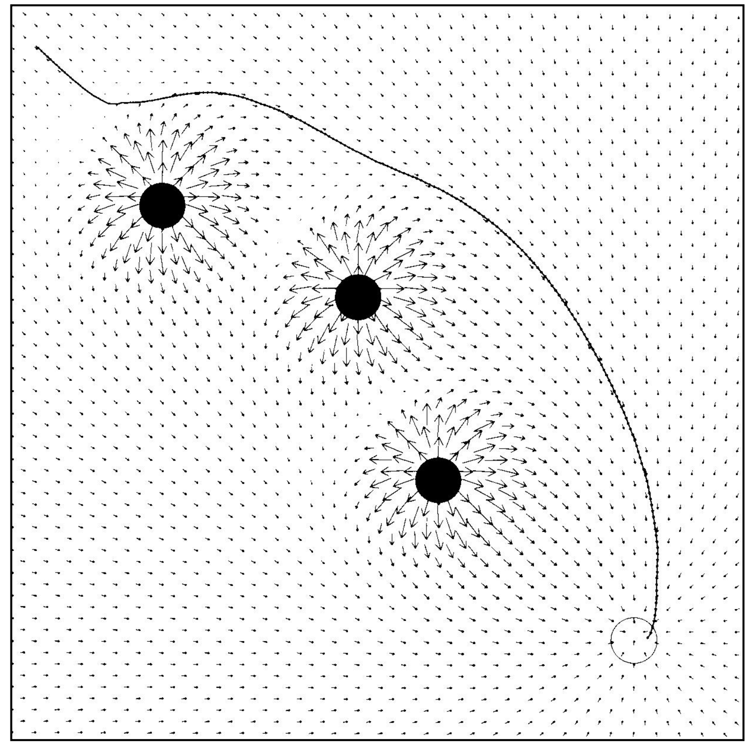
Behaviour fusion via vector summation

9.4 Behaviour Assembling

Behaviour fusion

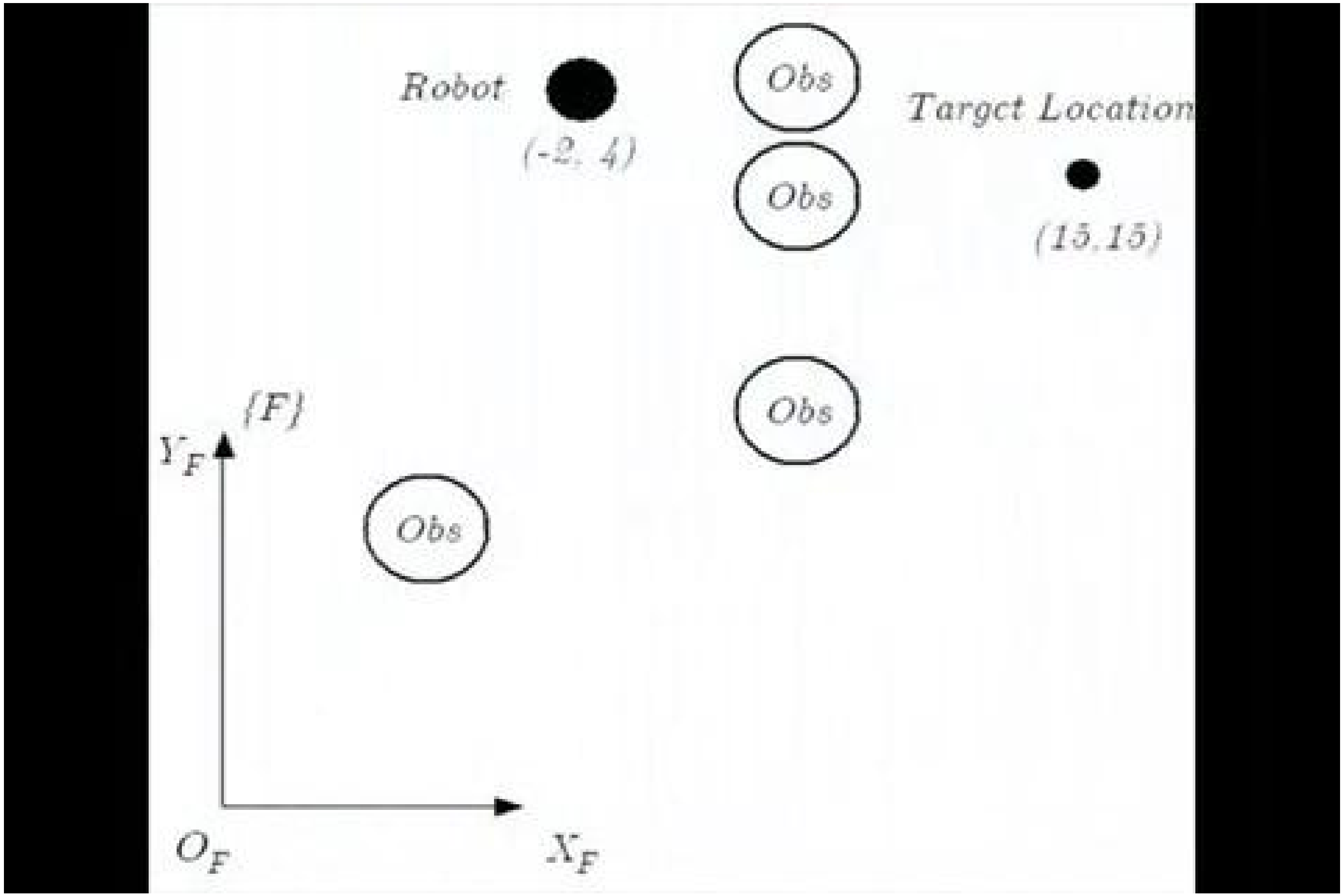


(a) goal attraction dominates



(b) obstacle avoidance dominates

9.4 Behaviour Assembling



9.5 Conclusions

- Robot behaviours generate a motor response from a given perceptual stimulus, and avoid the use of explicit representational knowledge.
- Reactive or behaviour-based systems are inherently modular in design based on biological models.

The four popular expressions of robot behaviours:

-- ***SR diagrams*** intuitively convey the flow of the control within a reactive system.

-- ***Function notation***: is a mathematical method to express robot behaviours

-- ***FSA diagrams***: are well suited for representing behaviour assemblages in time-varying composition.

-- ***Formal methods***: include RS models and SA models.

9.5 Conclusions

- Robot behaviours are encoded in two forms:
  - -- ***discrete encoding***: rule-based methods are often used.
  - -- ***continuous encoding***: the potential-field methods are often used.

- The two primary methods for behavioural coordination are competitive and cooperative. They can be combined if necessary.
  - -- ***Competitive methods***: result in the selection of the output of a single behaviour, typically either by arbitration or action-selection.
  - -- ***Cooperative methods***: often use super-positioning of forces or gradients generated from field-based methods, e.g. potential fields.

9.5 Conclusions