Using finite state machine in robotics

Wouter Lieffijn, Leendert van Achteren

Groningen, november 2005

1 Introduction

Using the term 'state machines', people often refer to finite state machines (FSMs). Besides finite state machines, also liquid state machines (LSM) exist. However, LSMs have nothing in common with finite state machines and will therefore not be regarded in this paper.

2 Working of finite state machines

Finite state machine are models of behavior composed of states, transitions, and actions[1].

- States store past information,
  i.e. they reflect input changes from the beginning to the present moment;
- Transitions indicate change of states. Transitions are described by a condition needed to be fulfilled to enable the transition;
- Actions may be performed at given times. Several action types exist:
  - Entry action: executed when entering the state;
  - Exit action: executed when exiting the state;
  - Input action: executed dependent on present state and input conditions;
  - Transition action: executed when a certain transition is performed.

3 Representation of finite state machines

Different representations of FSMs exist. Two well-known representations are state (transition) diagrams (Fig. 1) and state (transition) tables.

The most common representation is shown in: the combination of current state (B) and condition (Y) shows the next state (C).

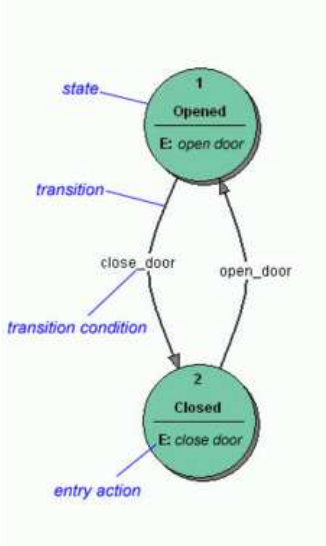

Fig 1: a state diagram

	State A	State B	State C
Condition X	...	...	...
Condition Y	...	State C	...
Condition Z	...	...	...

Fig 2: a state table

4 Classification of FSMs

Two types of FSMs are distinguished: acceptors/recognizers and transducers.

4.1 Acceptors/recognizers

Acceptors/recognizers may accept or reject a given input. These FSMs define a certain language or grammar to which input must meet. States of acceptors are said to be either accepting or rejecting. If the acceptor has processed the complete input, it accepts the input. If the acceptor aborts the process, the input is rejected. Often, the input is represented as characters. Acceptors do not use any actions.

Fig 3 shows an acceptor solely accepting the word 'nice'. So the only accepting state is the 7th state.

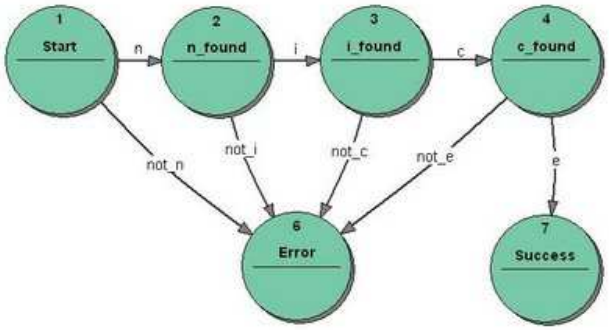

Fig 3: an acceptor

4.2 Transducers

Transducers generate output based on a given input and/or a state, using actions. Transducers are often used for controlling applications. Two subtypes of transducers are distinguished: Moore-transducers and Mealy-transducers.

4.2.1 Moore-transducers

A Moore-transducer uses entry actions, i.e. output depends on both input and the current state. Fig 5 shows a Moore-transducer for an elevator door recognizing the commands 'command_open' and 'command_close' that trigger a state change. The entry action (E:) in state 'Opening' starts a motor to open the door, the entry action in state 'Closing' starts a motor to close the door. States 'Opened' and 'Closed' themselves do not perform any actions. They just signal to the outside world (e.g. to other FSMs) the situations 'door is open' or 'door is closed'.

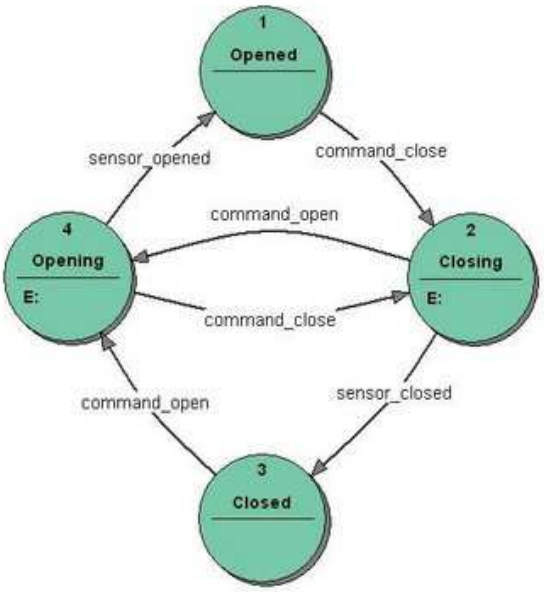

Fig 4: a Moore-transducer

4.2.2 Mealy-transducers

A Mealy-transducer uses only input actions, i.e. output depends only on input, not on the current state. Advantage is the simplification of behavior[2]: use of Mealy-transducers often leads to reduction of the amount of states. Fig 5 shows a Mealy-transducer implementing the same behavior as the Moore-transducer from Fig 4. Two input actions (I:) exist:

- 'start motor to close the door if "command_close" arrives'
- 'start motor to open the door if "command_open" arrives'.

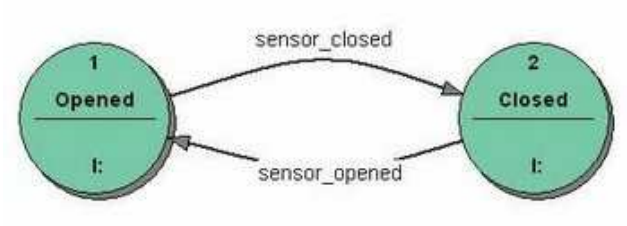

Fig 5: a Mealy-transducer

4.3 Further distinctions

Another distinction is between deterministic and non-deterministic FSMs. In deterministic FSMs, for each state there is exactly one transition for each possible input. In non-deterministic FSMs, in one single state there may be no or more than one transitions available for each possible input.

FSMs containing only one state are called combinatorial, using only input actions. Combinatorial FSMs are useful when different FSMs must cooperate or when it is convenient to consider a purely combinatorial part as a form of FSM to suit the design tools.

5 Examples

5.1 Virtual/Software FSMs

Video game developers make extensive use of FSMs[3]. Since ID Software released the source code from Quake and Quake 2, users might have noticed that movement, defense, and offense of the bots were controlled by simple FSMs. ID is certainly not the only company to take advantage FSMs. Games like Warcraft III make use of very complex FSMs for AI-control. FSMs may also run chat dialogs, prompting available choices for a user.

One might say that websites use FSMs, offering menus to traverse detailed sections of the website.

5.2 Physical/Hardware FSMs

Physical FSMs perform a great role in cars, airplanes, and robotics.

Snuf[4] is a simple autonomous mobile robot that gathers small blocks from the floor and brings them to a predefined position, avoiding other objects. Besides a smooth floor, no further operating requirements are needed.

One FSM is used for goal specification, while another FSM is used for object avoidance. This approach allows easy extension of functionality and features learning and probabilistic behavior.

Fig 6: Snuf

The goal state diagram (Fig 7) describes how the state machine can make a transition from one state to another and what event triggers such a transition.

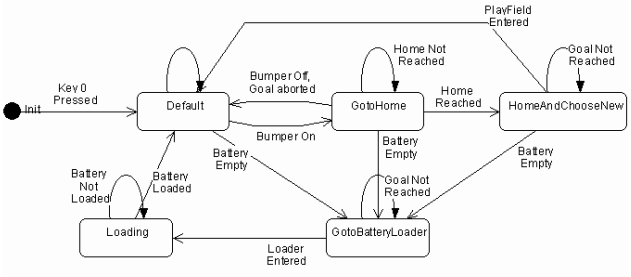

Fig 7: a simplified diagram of Snuf's goal FSM.

Three states ('Default', 'GotoHome' and 'HomeAndChooseNew') define Snuf's general behavior; two others ('GotoBatteryLoader' and 'Loading') are implemented for automatic battery charging. Each of these states defines a different goal to achieve. Fig 8 summarizes these goals.

state	goal to achieve
Default	search for a block
GotoHome	take home block
HomeAndChooseNew	return to field
GotoBatteryLoader	go to battery loader
Loading	wait and charge

Fig 8: Snuf's states and goals

Most important are the search for blocks in the default state and 'homing' of a captured block in the GotoHome-state. Snuf starts looking for a block and tries to bring a found block to its home. Fig 9 depicts a typical instance of a sequence of state transitions.

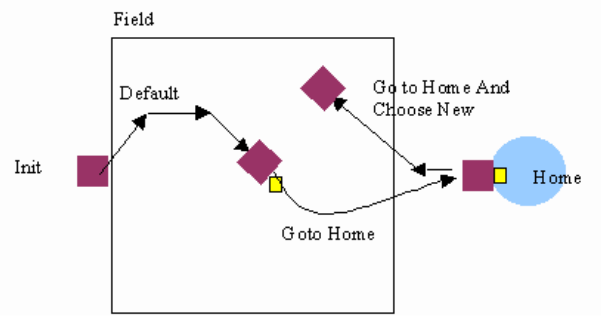

Fig 9: a typical sequence of state transitions

Some Goal-Aborted-event might put Snuf from the GotoHome-state to the default state. The Echo-locator typically emits such a Goal-Aborted event. Snuf's engine empties the command stack to enter emergency commands for escaping an object.

6 Discussion

While technically not being very advanced, FSMs still appear to be useful in applications requiring limited amounts of states. For complex applications however, FSMs are not suitable because the amount of states necessary, slows down the application too much.

---

1 http://en.wikipedia.org/wiki/Event_driven_finite_state_machine

2 Wagner, F. (2005) *Moore or Mealy model?* http://www.stateworks.com/active/download/TN10-Moore-Or-Mealy-Model.pdf

3 http://www.generation5.org/content/2003/FSM_Tutorial.asp

4 http://home.planet.nl/~j.havinga/snuf/snuf.htm