A Model for Transition-Based Visuospatial Pattern Recognition

by

Telmo Luis Correa Junior

B.S., Massachusetts Institute of Technology (2009)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Masters of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2011

Author ....[illegible]........................
Department of Electrical Engineering and Computer Science
May 24, 2011

Certified by ....[illegible]........................
Patrick H. Winston
Ford Professor of Artificial Intelligence and Computer Science
Thesis Supervisor

Accepted by ....[illegible]....................
Dr. Christopher J. Terman
Chairman, Masters of Engineering Thesis Committee

A Model for Transition-Based Visuospatial Pattern Recognition

by

Telmo Luis Correa Junior

Submitted to the Department of Electrical Engineering and Computer Science
on May 24, 2011, in partial fulfillment of the
requirements for the degree of
Masters of Engineering in Electrical Engineering and Computer Science

Abstract

In my research, I designed and implemented a system for learning and recognizing visual actions based on state transitions. I recorded three training videos of each of 16 actions (approach, bounce, carry, catch, collide, drop, fly over, follow, give, hit, jump, pick, push, put, take, throw), each lasting 10 seconds and 300 frames. After using a prototype system developed by Dr. Satyajit Rao for focus and actor recognition, actions are represented as qualitative state transitions, tied together to form tens of thousands of patterns, which are then available as action classifiers. The resulting system was able to build simple, intuitive classifiers that fit the training data perfectly.

Thesis Supervisor: Patrick H. Winston
Title: Ford Professor of Artificial Intelligence and Computer Science

Acknowledgments

I thank first my parents, for supporting me pursuing my career and my dreams at every step, even when it meant moving abroad.

I thank Dr. Rao and Professor Winston for the advice received during this research project, and for the opportunity to do this research work.

I thank my friends, at MIT and elsewhere, for their company and support.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This first chapter introduces an overview of the visuospatial action recognition problem, breaking it down into smaller problems, and making explicit which of those problems my research focused on.

Chapter 2 discusses previous research done in this framework on visuospatial action recognition.

Chapter 3 introduces the transition model, and goes over initial experiments.

Chapter 4 details boosting classifiers, and the central results of my research.

Chapter 5 summarizes the findings and contributions of this research.

1.1 Overview

A visuospatial recognition system, subject of my research, is responsible for extracting and recognizing actions in captured video. The system's task can be summarized as follows:

- Capture video input
- Extract frame information
  - Identify actors
  - Select focus

  - Acquire information about current focus and its relation to other actors

- Modify representation to make relevant elements more explicit

- Learn actions

- Use learned information / test the system

Each of those tasks relies on the output of the previous task; nonetheless, the dependencies are modular, in the sense that only the requirements at each step are the format and contents of the output. Another actor recognition algorithm could be used, as long as it produced the same format of output; other filtering could be done on that output.

1.1.1 Video capture

Training samples consisted, throughout this and previous research, of 3 sets of videos for visuospatial recognition: vsr1, vsr2, and vsr3. vsr2 was the first recorded set – the original set of recorded videos was deemed poorly recorded, and never used. vsr3 was recorded later, taking into account some characteristics of the program being used to extract frame information, trying to make events cleaner and contact more explicit; both vsr2 and vsr3 were used by previous work. I recorded vsr1 myself, wishing to have yet another set of videos on which to test the data.

Each of the training sets consists of 48 or more videos, 3 for each of the 16 actions my research attempts to identify: approach, bounce, carry, catch, collide, drop, fly over, follow, give, hit, jump, pick, push, put, take, and throw. vsr2 also includes some extra videos for a few of the actions; those extra videos were ignored on my research.

Video samples were recorded as colored video, at 320 x 240 resolution, at a rate of 30 frames per second. Color and image size are only relevant for the actor recognition stage of the system; for this work, the placeholder component recognizes actors using colors, and the subjects in the videos wear colored shirts. The frame rate, on the other hand, affects future steps of the process: the granularity of the frames is related to timing considerations – a limit distance of 60 frames, for example, is a timing limit of 2 seconds between events.

1.1.2 Extract frame information

The processing of information from each frame of video itself was done by previous research, and its implementation, BLOBS, detailed in Chapter 2. At first, reference frame information is produced for each frame, using the information available from the frame stream so far (figure 2-2); it contains information about which actors are appear in the corresponding frame, and their relative positions.

1.1.3 Later stages

The remaining stages of the visuospatial action recognition system were the focus of my research. Chapter 2 will explain previous work used for this part of the process, in particular filtering qualitative descriptions out of reference frames, via ATT, and a previous translation of that qualitative description into hardcoded actions, via RFTELL. RFTELL was also responsible for providing inspiration for my transition-based models, discussed in chapter 3 and also used in chapter 4.

Chapter 2

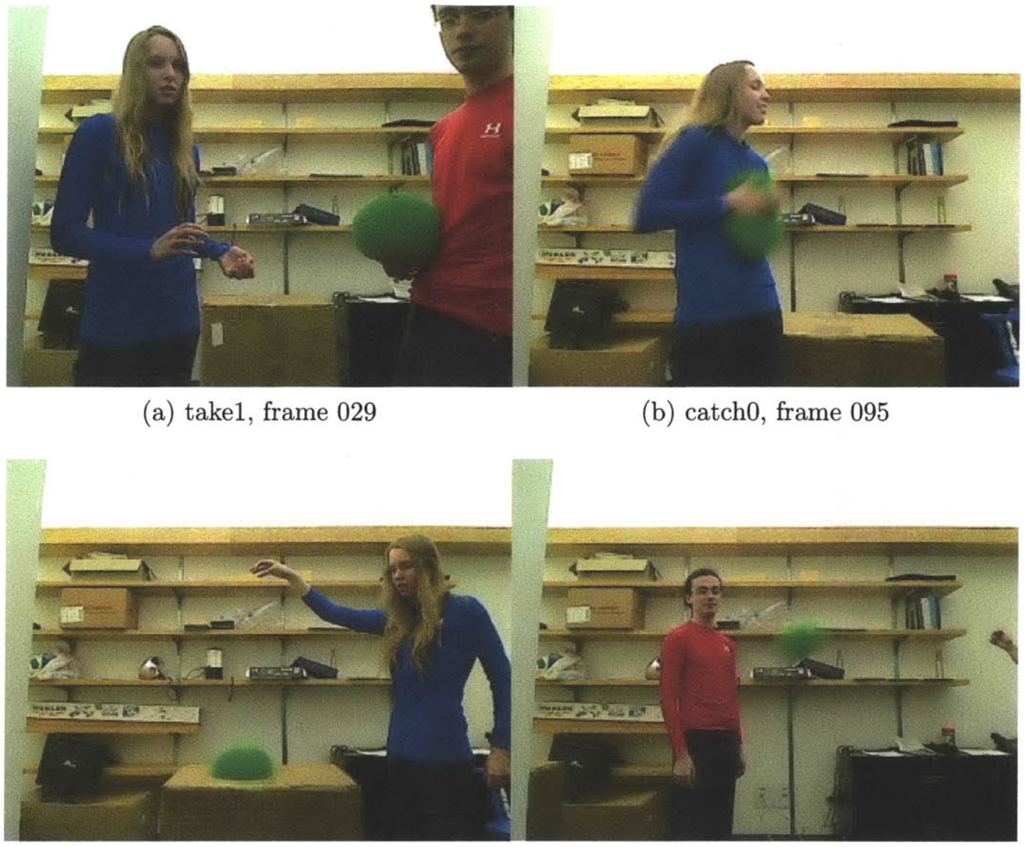
Previous work

This chapter discusses previous work done in this framework for visuospatial action recognition. It describes BLOBS and ATT, components of the system used previously and still used for my final results; it mentions previous work in recognition, and details RFTELL, a demonstration of action recognition that contains key ideas for my research.

When done with this chapter, the reader will be familiar with early stages of the system, actor and focus recognition, what information is received by the components I wrote for my research, and the motivations for having created a new representation.

The initial parts of the system, focus and actor recognition, and filtering, were implemented by Dr. Rao as the BLOBS and ATT programs, and are not the focus of my research. They are used in the initial stages of the visuospatial action recognition system, past video capture, and detailed in this chapter.

This chapter will also briefly touch on two previous approaches for the later recognition stages of the system: using hidden markov models, and using hardcoded transition rules.


(a) take1, frame 029 (b) catch0, frame 095

(c) bounce1, frame 025 (d) hit1, frame 041

Figure 2-1: Sample captured frames

2.1 Blobs

Blobs is responsible for processing video frames, as png files (see 2-1) and producing reference frames for each of them, based on the stream seen so far (figure 2-2). In the process, it does actor and focus recognition.

Blobs was implemented as a prototype for the initial stage of action recognition; it supports at most 3 actors on a scene, who must be color coded with configurable colors (usually red, blue, and green, for best detection). Nonetheless, it supports real world video in real time. Current research by Dr. Rao is working on the harder problem of actor recognition and segmentation in real scenes, without the aid of color-coding.

2.1.1 Actors

An actor is a point of interest on the scene, that tends to change almost continuously with time.

The implementation used for this stage, BLOBS, uses calibrated colors to identify up to three actors in a scene. BLOBS was implemented by Dr. Rao [Rao, 1998] and used throughout this research, because my focus of it is not on the actor recognition problem itself.

Two main issues are relevant about actor recognition: first, only moving objects can currently be identified as actors. Other points in the scene could be of interest: a point around which other objects orbit, or even a background, for example. Occlusion could also be an issue here: ideally, this stage of the system should be able to keep tracking an actor as it goes behind an obstacle.

The second issue, segmentation, becomes more evident given with more complex actors. Since BLOBS represents a person as a single actor, information is never captured, such as the relative arm position to the body, or what part of the body made contact with another actor.

2.1.2 Focus

Focus selection [Rao, 1998] is also handled by BLOBS. At any point in time, the system can be focused on one of the actors, or nowhere, if there are no actors in scene. The criteria used for focus change is simply assigning focus to the fastest moving actor; this part of the system could be informed by higher stages of the cognition to instead pay attention to or look for a specific actor, in future work.

2.1.3 Actor recognition

The raw output from the first stage of video processing, done by BLOBS, produces a reference frame for each frame of video. This reference frame contains very precise information, such as location of the current focus on the frame, its current vertical and horizontal speed, the relative location of other actors with relation to it in the frame, and contact flags on 12 angle sectors. Each of the frames, called a reference frame, is built as the sample in figure 2-2.

```
frameid: 4
nm=3 foaid=1 foaSize=0.044219 tracking=1 dx=0.000000 dy=0.005000 R=0.005000 Theta=1.570796
[0] R=0.405123 Theta=-0.024686
[1] R=0.000000 Theta=0.000000
[2] R=0.300666 Theta=-0.066568
ContactIds[12]= -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1
```

Figure 2-2: Sample reference frame produced by BLOBS

The first line identifies the video frame from which this reference frame was constructed. The second line contains the following variables described in table 2.1.

Table 2.1: Definitions for reference frame

field	description
nm	number of actors recognized in the scene
foaid	id of the actor currently on focus, from 0 to nm - 1, or -1 if none
foaSize	size of the actor currently focused
tracking	whether the actor is being tracked (1 if yes, 0 otherwise)
dx, dy	actor's movement in the horizontal and vertical axes since last frame
R, Theta	actor's position in polar coordinates

The following lines contain the position in the scene for each of the actors, in polar coordinates, relative to the actor currently being focused on. If the actor is not in the scene, the line referring to it does not appear.

The last line indicates whether the actor currently in focus is in contact with other actors. For each of 12 arcs of 30 degrees, the values can be -1 is there is no contact on that arc, or an actor id if there is contact with that actor.

2.2 Att

The backbone of the system is the previous work done by Dr. Rao for processing a video stream into attention traces. The key idea is that it is sufficient for a system to be focused at only one actor of a scene at a time, the properties of that actor, and its relations to other actor. This approach greatly reduces the complexity of the data, makes the system more scalable to a larger number of actors.

2.2.1 Attention frames

Reference frames then have their information filtered into *attention traces*, by ATT. The finely-quantified, almost continuous information in reference frames is transformed into discrete changes; significant changes can be detected and assigned to a specific frame – as soon as the change goes over a detection threshold. A fragment of an attention trace is shown in figure 2-3.

```
Inst 49 [- 48 47 46 45] id=2  ddx=-1 [0] dR=1 dTheta=0
KF 49 [- 35] id=2  [0] dR=1 dTheta=0
Inst 50 [- 49 48 47 46] id=2  ddx=-1 [0] dR=1 dTheta=0
KF 51 [- 11] id=1  [2] dR=-1 dTheta=0
Inst 55 [- 54 50 49 48] id=2  [0] dR=1 dTheta=0 [1] dR=-1 dTheta=0
Inst 57 [- 56 53 52 51] id=1  [2] dR=-1 dTheta=0
KF 58 [- 26] id=0  [2] dR=1 dTheta=1
KF 59 [- 49] id=2  dmarkerContact[1]=1
Inst 60 [- 59 55 54 50] id=2  dmarkerContact[1]=1
Inst 61 [- 60 59 55 54] id=2  [1] dR=0 dTheta=1 dmarkerContact[1]=1
```

Figure 2-3: Sample fragment of attention frame log, produced by ATT

Any meaningful changes, after filtering, are assigned to specific frames. Each of those frames can be of one of two types: *instantaneous frames*, and *key frames.*

Changes detected and assigned to an *instantaneous frame* represent a continuous change, detected via ATT's thresholds to have started on that frame – like an actor moving.

Changes detected and assigned to a *key frame*, on the other hand, represent the change of a trend: objects that were moving in one direction stopped, or started moving in another direction, or contact between two objects is broken. Large changes in an event can be entirely described by the key frames.

Using both instantaneous frames and key frames provides a description of all changes deemed significant enough by the filtering in ATT. Using only key frames provides a meaningful description of the event as a whole, because key frames include only the most significant changes in physical state.

Note that the mapping between reference frames and instantaneous or key frames is not one-to-one. A reference frame may not be mapped to either, if there are no changes happening – a reference frame cannot be taken out of context here, because this processing depends on change across a number of frames. Similarly, one instantaneous and one key frame may be produced for a same frame number.

Note also that, due to the nature of filtering and discretization of change, very small or slow changes are considered to be noise, and ignored. The exact thresholds are calibrated manually to detect changes at the correct scale.

Note, finally, that while reference frames include floating point numbers for position and movement, this representation reports only the change on the vertical and horizontal axis. Information about trajectory is sacrificed: there is no notion of an angle of movement, only up and down, left and right, and the combinations of those.

2.2.2 Actions

The concept of a visuospatial action, or pattern, needs to be somehow defined. Concepts represent to something one might assign a verb to, such as *give* or *jump*, or a more complex, specific description, such as *hit from the left, then break contact and fall off the scene.*

With supervised learning, the system should be able to infer something common about all videos reprenting the same action, and extract a model (or models) that can be used to recognize the action later on. With unsupervised learning, such models must be extracted from similarity between unlabeled actions. In my research I proposed such models, how are they extracted, learned, and how they can be used.

2.3 Previous work on action recognition

2.3.1 HMMs

Prior to the research reported in this thesis, only supervised learning was used: a Hidden Markov Model (HMM) was extracted from each group of labeled actions, using a human entered guess as to the number of states. Once a HMM was produced for each action, a meta HMM, (i.e., a HMM using the state of the others as input) was used to try to identify which of the actions was occurring at any given time, being responsible for both recognition and prediction.

The system worked, but presented some weaknesses. In particular, the extraction of HMMs does not generalize easily to unsupervised learning, without the use of some non-intuitive similarity measure, and the system was particularly weak with regard to timing variations. This led me to start looking at another model for representation.

2.3.2 RFTell

RFTELL, a perl script, was put together in previous research to show initial parts of the system (BLOBS, ATT) working, and demonstrate the potential for visual action recognition in real time. It processes reference frames and producing a description of the physical state of the system. Such description had a small number of variables, listed in table 2.2, each of which could take only one of a very small number of values, listed in the same table. Hand-encoded rules would check for those variables changing, or remaining constant, over time, and trigger whenever completed. A triggered rule would send output to a text-to-speech program (festival), providing audio to be put along with the video for a demonstration.

A rule would then be handcoded as a sequence of variables changing, or not changing. Some of the most complex rules are listed in figure 2-4.

For "A1 picked up A0," at first Y_0 changed from 0 to 1, while C_{01} is 0; that is, actor 0 starts moving up while not in contact with actor 1. Then, C_{01} changes from 0 to 1; that is, contact is detected between actors 0 and 1.

Table 2.2: Definitions of state variables in RFTELL

variable	description
foaid	-1 if not focused anywhere, the actor ID that is the current focus otherwise.
E_i	1 if actor i is in scene, 0 otherwise.
M_i	1 if actor i is moving, 0 otherwise.
X_i	1 if actor i is going to the right, -1 if to the left, 0 if not moving horizontally.
Y_i	1 if actor i is moving up, -1 if moving down, 0 if not moving vertically.
S_i	1 if actor i is getting bigger, -1 if shrinking, 0 otherwise.
C_{ij}	1 if actors i and j are in contact, 0 if not.
R_{ij}	1 if the distance between actors i and j is increasing, -1 if decreasing, 0 otherwise.
T_{ij}	0 if actor j is to the right of actor i, 1 if to the top, 2 if to the left, 3 if to the bottom.

```
1000    C0,1=1,0;;Y0=0,-1;;C0,1=1,0              A1 dropped A0
2000    T0,1=0,3;C0,1=0;;T0,1=3,2;C0,1=0          A0 flew over A1
2000    T0,1=2,3;C0,1=0;;T0,1=3,0;C0,1=0          A0 flew over A1
3000    C0,1=1,0;C0,2=0;;C0,2=0,1;C0,1=0          A1 gave A0 to A2
3000    C0,2=0,1;C0,1=1;;C0,1=1,0;C0,2=1          A1 gave A0 to A2
2500    Y0=0,1;C0,1=0;;C0,1=0,1                   A1 picked up A0
```

Figure 2-4: RFTELL hardcoded sample rules

A numerical priority is associated with each rule, because some actions are usually detected as a subset of other actions.

The structure of the rules themselves is the most important result from RFTELL: some variables are required to change, while other variables are required to stay constant, but most state variables are irrelevant. The picking-up action should not depend on the presence of a third actor in the scene, for instance.

Note also that one same action may have more than one possible encoding: in the sample above, flying over is triggered both for starting at the left quadrant and moving to the top quadrant, or starting at the right quadrant and moving to the top quadrant. Those actions are distinct, from a visuospatial view (one could conceivably be called *flying over from the left* and the other *flying over from the right*), but both are grouped under the same description.

Note, finally, that the same action could conceivably originate from two different rules, depending on the current focus and the order with which the changes are detected. For the sample of *give*, there are two rules, one where the receiver makes contact with the object and then the object breaks contact with the giver, and one where the object breaks contact with the giver before it makes contact with the receiver. Even given a large enough time lapse to make those sequences distinct (in which case it becomes clear that throwing an object to someone is one instance of give), both rules have the same start and end state.

Chapter 3

Transition-based representations

This chapter introduces the transition-based representations: why they were selected, how they compare with previous work, and how they were modified during my research (section 3.1). It explains how transitions are put together in patterns (section 3.2), and how those can be used to identify actions.

This chapter also goes over experiments in learning actions, extracting patterns from sample files and using them for recognition, as simple subsequences (subsection 3.3.1), and then using hits and misses (subsection 3.3.2). Those experiments were unsuccessful in producing real-time results, consistently identifying an intuitive description for actions, and recognizing actions themselves.

Experiments with unsupervised learning, using patterns and a specific data structure, are included in subsection 3.3.3. Those experiments produce a better-than-random, but ultimately unsatisfactory, prediction rate for transitions.

Section 3.4 includes suggestions on how to modify the present detection model for future work; this research direction was not pursued, in favor of the approach taken in chapter 4.

When done with this chapter, the reader be familiar with the transition-based representation used in my research, its inspirations, and its uses early unsucessful experiments.

Following a suggestion by Professor Winston, I looked into work by Borchardt about discrete representations for states and transition [Borchardt, 1984, 1993]. RFTELL already used a representation very similar to the one I wanted to achieve; adopting and modifying it swas a natural implementation choice.

3.1 Transitions

3.1.1 First model: state representation and transitions

The first attempt at a model was to make explicit the framework under which such variables existed. The concept of a *state representation* was created: at each point in time, the physical state could be concisely summarized by assigning a value to each of the variables. Following a suggestion by Dr. Rao, I made the focus variable explicit; while RFTELL did use it to track the rule triggered when the focus was in no actors, it was unused otherwise; it would be interesting to see whether the actual focus had any bearing on obtaining a good description of actions. Each state, then, contained the following variables, essentially identical to the representation used for RFTELL (table 3.1).

Table 3.1: Updated definitions from RFTELL

variable	description
F	**-1 if not focused anywhere, the actor ID that is the current focus otherwise.**
E_i	1 if actor i is in scene, 0 otherwise.
M_i	1 if actor i is moving, 0 otherwise.
X_i	1 if actor i is going to the right, -1 if to the left, 0 if not moving horizontally.
Y_i	1 if actor i is moving up, -1 if moving down, 0 if not moving vertically.
S_i	1 if actor i is getting bigger, -1 if shrinking, 0 otherwise.
C_{ij}	1 if actors i and j are in contact, 0 if not.
R_{ij}	1 if the distance between actors i and j is increasing, -1 if decreasing, 0 otherwise.
T_{ij}	0 if actor j is to the right of actor i, 1 if to the top, 2 if to the left, 3 if to the bottom.

A change one variable between two consecutive states was deemed a *transition*. Transitions were deemed the most important part of this representation; ideally, any action could be represented as unambiguously as a sequence of transitions. Each transition would consist of a variable type, its corresponding actors, a previous state and a new state. For example, $C_{12} = 0 : 1$, or $E_2 = 0 : 1$.

The representation used in RFTELL is closer to Borchardt's, in that it tracks both changes in state variables, and lack of change: as seen in rule given for "A1 picked up A0", in figure 2-4, a state variable could change values (Y0 changing from 0 to 1) or keep a constant value (C0,1 staying constant at 0). Transitions, on the other hand, only track change of value; they have the general format $V = A, B$, for some state variable V , and before and after states A and B.

Another notable difference between this representation and the previous one is that the focus is now explicit: rather than mere consideration, the F variable becomes part of the description of the scene, and could be used for modeling actions.

Transitions could be extended encompass unknown states: the special value of "?", could be used in the previous or new state fields to represent that the current state was unknown, such as when starting the system, or not applicable, such as the contact between objects 0 and 1 when object 1 was not in the scene.

Extending transitions to work as masks was just as simple: add a wildcard character, "*", to represent that any value would be accepted on that field.

After implementing with this framework, I proceeded to look at the attention trace itself.

Event traces

I envisioned the transitions outlined above being used in the next stage of the system: each event gets transformed into a sequence of transitions, each with an associated frame number from its occurrence, and this stream of timestamped transitions is fed into the next stage of the system.

Again inspired by the RFTELL representation, I wrote a program, REP, to translates a stream of reference frames into a stream of transitions, using exactly the same filters of RFTELL. A fragment of the stream is shown in figure 3-1.

REP did some filtering similar, but not identical, to the one used in ATT. Since the filtering done by ATT produced results in previous work, I should be using the exact same filtering to obtain the same output, and focus only on the machine learning parts of the process. It should produce the information in this new transition-based

```
43      F       0       2
43      T1,2    3       2
44      F       2       1
44      T1,2    2       3
45      F       1       2
45      T1,2    3       2
68      R0,2    0       -1
69      R1,2    0       1
70      C0,2    0       1
71      X2      0       -1
71      M2      0       1
79      C1,2    1       0
```

Figure 3-1: Event stream fragment produced by REP (training set vsr2, give0, frames 43 - 79)

representation, however.

For this purpose, I created another program, ATT2REP, that converts the output of ATT into a stream of transitions and frame numbers, as envisioned before and produced by REP.

The representation from ATT, however, did not contain any reference to quadrants; instead, it contained information about whether the angle between two actors was changing. To adjust for this, I had to expand the state representation and transition models to include a new type of variable, W, to represent this change: initially W_{ij} would assume the value 1 for clockwise rotation, -1 for counterclockwise, and 0 for insignificant rotation of the actors around each other.

Changes in quadrant (T) were not to be detected anymore. The quadrant borders at 45 degrees were somewhat arbitrary, and had been proven brittle in initial testing – small rotation inside a quadrant would not be detected, but some rotation noise around the diagonals would survive filtering. The variable type was still kept in place for backwards compatibility, but not produced by ATT2REP.

After initial testing, no actions were found where handedness was relevant. All actions tested could happen "from the right" or "from the left" under the same label, and patterns where W variables did assume both 1 and -1 values were not found to be relevant to the oberved action itself.

In order to keep the model simpler, I fixed the definition of W variables to track only whether two objects are rotating around each other, not the direction of rotation is clockwise or counterclockwise. If required, undoing this at later stage should be simple enough, but that never became necessary.

The new complete list of variable descriptions is shown in table 3.2.

Table 3.2: Definitions of state variables derived from RFTELL, final version

variable	description
F	Current focus. -1 is none, or actor number otherwise.
E_i	1 if actor i is in scene, 0 otherwise.
M_i	1 if actor i is moving, 0 otherwise.
X_i	1 if actor i is going to the right, -1 if to the left, 0 if not moving horizontally.
Y_i	1 if actor i is moving up, -1 if moving down, 0 if not moving vertically.
S_i	1 if actor i is getting bigger, -1 if shrinking, 0 otherwise.
C_{ij}	1 if actors i and j are in contact, 0 if not.
R_{ij}	1 if the distance between actors i and j is increasing, -1 if decreasing, 0 otherwise.
W_{ij}	**1 if actors i and j are rotating around each other, 0 otherwise.**
T_{ij}	0 if actor j is to the right of actor i, 1 if to the top, 2 if to the left, 3 if to the bottom.

The change in this representation is the inclusion of the new variables, W_{ij}, to represent ongoing change of relative orientation. As noted above, this representation is a superset of the previous representation: no matter whether only T_{ij}, W_{ij} or both types of variable are fed in the stream, both types of change can be captured.

The same video fragment now produces the results seen in figures 3-2 and 3-3.

3.2 Patterns

With this framework in place, I still needed a model to represent the actions themselves. I started with the simplest possible concept: I defined a *pattern* to be a sequence of transitions, without unknowns, masks, or constant transitions. In order for a pattern to match with an event trace stream:

- All transitions listed on the pattern must occur on the event trace, in the listed order;
- Two transitions that have been *matched* to the pattern may not occur more than 15 frames apart;

```
47      F       2       1
47      R0,2    -1      0
47      W1,2    0       1
55      F       1       2
55      R0,2    0       -1
55      W1,2    1       0
56      X2      0       -1
56      M2      0       1
56      R1,2    0       1
57      X2      -1      0
57      M2      1       0
57      R0,2    -1      0
57      R1,2    1       0
66      R0,2    0       -1
66      W1,2    0       1
67      X2      0 -1
67      Y2      0       1
67      M2      0       1
67      R1,2    0       1
69      W1,2    1       0
70      C0,2    0       1
71      R1,2    0       1
74      Y2      1       0
75      W1,2    0       1
79      Y2      0       1
79      C1,2    1       0
```

Figure 3-2: Event stream fragment using ATT and ATT2REP (training set vsr2, give0, frames 43 - 79, using instantaneous and key frames from ATT)

```
67      F       -1      2
67      E0      0       1
67      X0      ?       0
67      Y0      ?       0
67      S0      ?       0
67      M0      ?       0
67      E2      0       1
67      X2      ?       0
67      Y2      ?       0
67      S2      ?       0
67      M2      ?       0
67      R0,2    0       -1
67      W0,2    ?       0
70      R0,2    -1      0
70      C0,2    0       1
77      R0,2    0       -1
79      E1      0       1
79      X1      ?       0
79      Y1      ?       0
79      S1      ?       0
79      M1      ?       0
79      R0,1    ?       0
79      W0,1    ?       0
79      R0,2    -1      0
79      R1,2    ?       0
79      W1,2    ?       0
79      C1,2    1       0
```

Figure 3-3: Event stream fragment using ATT and ATT2REP (training set vsr2, give0, frames 43 - 79, using only key frames from ATT)

- If two consecutive transitions A and B in the pattern refer to the same variable α, then no other transitions may occur in the event trace to the same variable α, between the transitions that match A and B.

The first two conditions, transition occurrence and matching limited to a small time window, help capture the concepts of sequential actions and locality. The last condition attempts to capture that allowing other types of change to occur does not detract from capturing a sequence of changes on a specific variable.

While these rules work for matching against instantaneous and key frames, they are bad for capturing key frames that occur too far from each other. So, when an event trace stream is produced using only key frames, the following rules for matching patterns were used instead:

- Before processing, discard all transitions with unknowns;
- All transitions listed on the pattern must occur on the event trace, in the listed order;
- No other transitions may occur between two transitions that match consecutive transitions in the pattern.

In effect, this makes the pattern be a substring of the (filtered) event trace, as opposed to a subsequence: in other words, the pattern for key frame matching must have all its transitions happening adjacent to each other, with no unmatched transitions in the event log in between.

3.2.1 Comparison with previous work

RFTell

The transition-only representation is not equivalent to the RFTELL representation: transitions as implemented cannot capture a state that remains constant while something else changes. The framework could be extended in future work to capture transitions with same before and after state, not mentioned in the event log, but this direction was not explored in the current work.

RFTELL also used its own filters, over a variable number of frames for each kind of scene variable. ATT, on the other hand, uses filters that always look at the past 4 frames to determine what is changing on the scene. **Different filters are used** for detecting qualitative changes. Different filters will produce slightly different results, even if they map to very similar representations; a comprehensive comparison of both representations was not explored.

HMMs

The previous work using HMMs for classification was particularly sensitive to timing: the timing of each action was captured from the training data, and performing the same action too slowly or too quickly would cause detection to fail. Timing information is not captured in my representation, other than the ordering of the transitions themselves. Accordingly, I expected this representation to be more robust to timing changes, as confirmed in experiments.

3.3 Experiments with the transition model

Using this transition-only model and concept of patterns, I did the first few experiments. First, I attempted to reproduce supervised learning from previous research.

3.3.1 Supervised learning

Representation used: transition-only patterns

Goal: verify that meaningful descriptions for each action can be extracted, when the system is presented with video samples for a single action type (for example, only *give* samples).

Method:

A first algorithm for finding subsequences across a limited number of input event trances was intuitive: simply search event logs for common subsequences, and build up larger patterns from smaller patterns. The same training set from the previous work, vsr2, was used here: 16 actions, with 3 recorded samples for each.

Patterns with 1, 2, 3 and 4 transitions were searched across the event traces for each individual verb of the training set vsr2, one set of 3 videos corresponding to a verb at a time. The results for the obtained number of patterns are below, as well as a representative description found for the verb.

Results:

Using both instantaneous and key frames: see table 3.3. Using only key frames: see table 3.4.

Table 3.3: Supervised learning experiment on vsr2, using instantaneous and key frames. The table shows the number of common patterns of 1, 2, 3 and 4 transitions, and a pattern selected by hand as representative of the action.

verb	# 1	# 2	# 3	# 4	description			
approach	4	7	8	8	$R_{01} = 0 : -1$	$R_{01} = -1 : 0$		
bounce	6	12	15	11	$Y_0 = 0 : -1$	$Y_0 = -1 : 0$	$Y_0 = 0 : 1$	$Y_0 = 1 : 0$
carry	6	17	45	99	$C_{01} = 0 : 1$	$M_0 = 0 : 1$	$M_0 = 1 : 0$	
catch	7	31	72	103	$R_{01} = 0 : -1$	$C_{01} = 0 : 1$	$Y_0 = -1 : 0$	
collide	11	44	112	176	$R_{01} = 0 : -1$	$C_{01} = 0 : 1$	$C_{01} = 1 : 0$	
drop	6	11	6	1	$Y_0 = 0 : -1$	$C_{01} = 1 : 0$		
fly_over	6	14	19	19	$W_{01} = 0 : 1$	$Y_0 = 0 : -1$		
follow	5	19	64	161	$M_0 = 0 : 1$	$M_0 = 1 : 0$		
give	10	96	548	2148	$R_{01} = 0 : -1$	$R_{02} = 0 : 1$	$C_{01} = 0 : 1$	$C_{02} = 1 : 0$
hit	10	55	183	376	$R_{01} = 0 : -1$	$C_{01} = 0 : 1$	$C_{01} = 1 : 0$	
jump	10	65	233	504	$Y_0 = 0 : 1$	$Y_0 = 1 : 0$	$Y_0 = 0 : -1$	$Y_0 = -1 : 0$
pick_up	7	33	121	362	$C_{01} = 0 : 1$	$Y_0 = 0 : 1$	$Y_0 = 1 : 0$	
push	8	48	113	154	$C_{01} = 0 : 1$	$C_{01} = 1 : 0$	$R_{01} = 0 : 1$	
put_down	8	41	157	462	$Y_0 = 0 : -1$	$Y_0 = -1 : 0$	$C_{01} = 1 : 0$	
take	10	130	1154	6786	$C_{01} = 0 : 1$	$C_{02} = 1 : 0$	$R_{12} = 0 : 1$	
throw	5	12	13	6	$C_{01} = 1 : 0$	$W_{01} = 0 : 1$	$Y_0 = 0 : -1$	

While the selected patterns provide an adequate enough description for the verbs, various issues were found with this approach, making the results unsatisfactory:

- The use of instantaneous and keyframes can produce too many common patterns

Table 3.4: Supervised learning experiment on vsr2, using key frames only. The table shows the number of common patterns of 1, 2, 3 and 4 transitions, and a pattern selected by hand as representative of the action.

verb	# 1	# 2	# 3	# 4	description
approach	1	0	0	0	$R_{01} = 0 : -1$
bounce	1	0	0	0	not detected
carry	0	0	0	0	not detected
catch	3	6	5	2	$R_{01} = 0 : -1$ $C_{01} = 0 : 1$ $R_{01} = -1 : 0$
collide	4	6	4	1	$R_{01} = 0 : -1$ $C_{01} = 0 : 1$ $R_{01} = -1 : 0$ $C_{01} = 1 : 0$
drop	2	0	0	0	$C_{01} = 1 : 0$
fly_over	1	0	0	0	$W_{01} = 0 : 1$
follow	0	0	0	0	not detected
give	7	12	10	3	$R_{01} = 0 : -1$ $R_{01} = -1 : 0$ $C_{01} = 0 : 1$ $C_{02} = 1 : 0$
hit*	5	8	7	3	$R_{01} = 0 : -1$ $R_{01} = -1 : 0$ $C_{01} = 0 : 1$ $C_{01} = 1 : 0$
jump	1	0	0	0	not detected
pick_up	2	1	0	0	$C_{01} = 0 : 1$ $R_{01} = 0 : -1$
push	3	2	0	0	$C_{01} = 1 : 0$ $R_{01} = 0 : 1$
put_down	3	2	0	0	$R_{01} = 1 : 0$ $C_{01} = 1 : 0$
take	6	30	102	195	$C_{01} = 0 : 1$ $C_{02} = 1 : 0$ $R_{12} = 0 : 1$
throw	2	0	0	0	not detected

on particularly verbose event traces; as seen in table 3.3, give and take have in the order of thousands detected patterns of size 4: not a good result for an algorithm aiming at selecting distinctive patterns by itself.

- Some actions did not produce a suitable description using only transitions from key frames. Some could, but the description was insufficient, such as the key frame descriptions selected for drop or fly_over.

- Some actions did not produce a suitable description at all. The only distinction between carry and follow, for instance, is whether the two objects are in contact during the whole action or not. Looking only at transitions fails to capture this element.

- The selection of a particular pattern, among others, as the representative for an action was made by hand – not an automated approach. A potentially useful heuristic was to pick the pattern with the most unusual combination of variable types; even then, the results would often disagree with a human hand-picked description for the pattern.

- The algorithm was unacceptably slow.
- The algorithm did not generalize directly to unsupervised learning.

3.3.2 Subsequences using hits and misses

Representation used: transition-only patterns

Goal: obtain meaningful descriptions of patterns, by using both positive samples and negative samples of an action as near-miss groups [Winston, 1992].

Method: similar method to the previous experiment in supervised learning, with the added requirement that patterns need to occur in every positive sample (hits), and in no negative samples of similar actions (near misses).

In particular, when examining the patterns from a new hit sample, only consider them if they have been seen on every other hit sample so far, and in no negative samples; when considering patterns from negative samples, remove them from the list of patterns considered so far, and add them to the set of patterns from negative samples.

The following combinations of similar verbs for hits and miss were tried: bounce and jump, hit and push, give and take, using one of the verbs of each group as a hit and the other as a near-miss.

Given the poor detection rates in the previous experiment, two of the conditions when extracting patterns were relaxed:

- The maximum distance between two consecutive patterns was increased to 60 frames, instead of 15.
- Any transition may be skipped between two matching transitions in an event log, both when extracting and detecting patterns. This is a change from the previous method, where a pattern would only be detected with consecutive

transitions on the same variable type α if no transitions of the same variable occurred in the file between those.

This experiment was performed after all other experiments in this chapter, and before the ideas that lead to the approach in chapter 4.

Results: tables 3.5, 3.6, and 3.7 detail the number of patterns obtained for each run of pattern extraction, for each of the 3 training sets (vsr1, vsr2, and vsr3). The first two columns display results using a verb as a hit, and no verbs as misses; the third and fourth columns display results using a verb as a hit and the other as a near miss.

The bounce recording in vsr3 includes multiple actors, besides the bouncing ball; the introduction of other actors (people watching the ball go) introduces many more transitions, producing the large numbers seen on the last row of table 3.5.

The results are unsatisfactory, for two reasons:

- The number of obtained patterns is too large to conceivably use this as a selection criteria for learning an action – at least with the size of the training datasets.

- The experiments often detected that some patterns were only present because of elements that appear to be irrelevant; for example, in the list of patterns of take without give, from vsr1, $X_0 = 0 : 1; R_{0,1} = -1 : 0; F = 0 : 1$ was detected, but $X_0 = 0 : 1; R_{0,1} = -1 : 0$ was not detected.

Nonetheless, some hints of structure are present: besides vsr3, the results for bounce without jump are empty, strongly implying that every jump video has bounce as an action also occurring on it – when the actor lands back on the floor.

3.3.3 Unsupervised learning

Representation used: transition-only patterns

Table 3.5: Supervised learning with hits and misses: jump versus bounce

	jump	bounce	jump w/o bounce	bounce w/o jump
vsr1	187	57	16	0
vsr2	377	15	132	0
vsr3	66	**1502**	14	**1454**

Table 3.6: Supervised learning with hits and misses: push versus hit

	push	hit	push w/o hit	hit w/o push
vsr1	523	108	348	19
vsr2	61	947	19	549
vsr3	271	114	213	59

Table 3.7: Supervised learning with hits and misses: give versus take

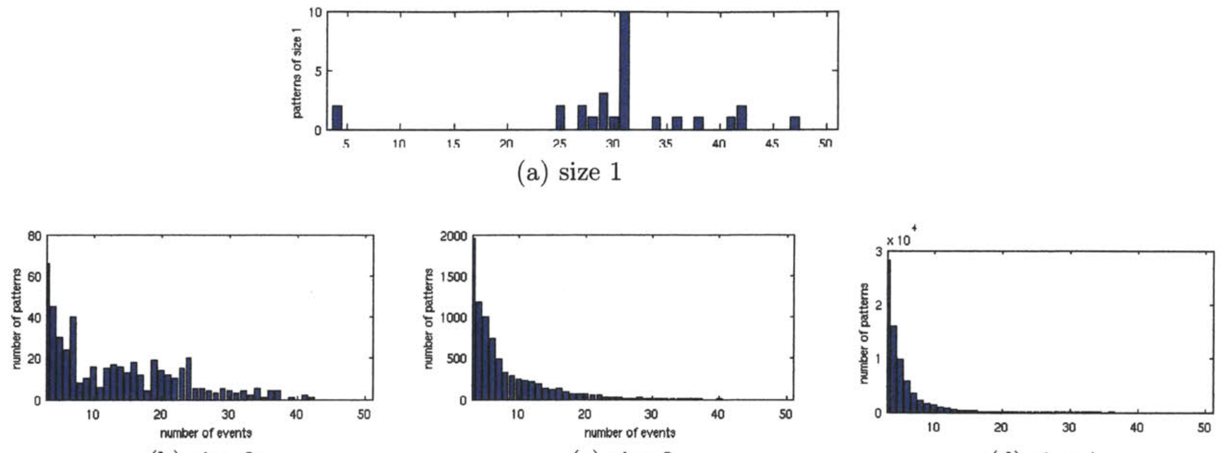
	give	take	give w/o take	take w/o give
vsr1	1597	587	567	123
vsr2	917	1906	223	737
vsr3	1368	526	613	166

Goal: verify that patterns extracted during supervised learning are still found. Verify that patterns found represent meaningful actions. Estimate size of problem.

Method and results:

The key idea for unsupervised learning was the same as for supervised learning: extract patterns that occur across multiple events. The difference now is that all event traces would be read together, without labels; a human might then label the obtained concepts after the process is done.

Initially the same algorithm from the supervised learning experiments was used, resulting in a running time of hours, and a result of thousands of patterns. All patterns picked as representatives for supervised learning were found – along with tens of thousands of others. In order to keep results manageable, only patterns that occurred in at least 3 training samples were captured. Histograms showing the number of patterns obtained this way can be seen in figure 3-4: each histogram shows the



(a) size 1 (b) size 2 (c) size 3 (d) size 4

Figure 3-4: Frequency of patterns in events by pattern size; see subsection 3.3.3

number of patterns of a given size (number of transitions), dividing them by frequency (number of sample events where they appeared as a subsequence at least once).

Weighted trie

To try to obtain an unsupervised learning algorithm with running time faster than a day, the naive intersect-and-increment algorithm was dropped. Taking inspiration from string matching algorithms, I decided to use a trie-like data structure; each node would contain a transition (instead of a character from a string) and have an associated weight. Each pattern would be represented as a path starting at the trie root, with one node for each of its transitions (as strings are represented in tries). Observed patterns from an input event log would be added to the data structure, creating new nodes or modifying weights of existing nodes; each pattern would contribute by increasing the weight of each of its nodes by 1. This way, the weight of a tree node would be the number of times the pattern starting at the root node and ending at that node was observed.

This trie-structure could be used for enumerating all patterns, and, in future work, for prediction. It could be modified to also keep track of which input events were responsible for each weight change, instead of just counting the number of events: actions could then be associated to specific node leafs.

All patterns could then be enumerated by searching through all nodes of the data structure and listing the patterns obtained from reading the transitions in the path from root to node.

Prediction could also be done from this data structure. Probabilities could be computed based on the frequency of the children of each node; doing so, however, would require an estimation of what node would correspond to the current state of the tree.

As a first, simple, approach, pointers were kept to all trie nodes corresponding to a partial pattern that matched the event log so far. Children were counted for each of the nodes indicated by the pointers, and weighed equally. This proved to be too slow, as then number of pointers quickly grew to the thousands, since pattern matching can skip transitions. The results, too, were less than ideal: both when using instantaneous and key frames, or only key frames, the probability of guessing the next transition correctly averaged around 7% – better than random, but unacceptably low. Other modifications could have been done to try to improve this result – such as restricting prediction only to a small subset of the trie leading to hand-picked patterns – but this line of research was stopped in favor of another, more promising model.

3.4 Future work: changes to modeling states

Both in the explicit results of supervised learning, and the implicit not-very-good performance of unsupervised learning, inadequacies of using only transitions were evident, such as the very low prediction rate. The representation used was not expressive enough to distinguish between some actions, using patterns as subsequences.

3.4.1 RFTell

The main flaw in the previous approach was not paying attention to state; as the results indicated, that model alone is not as descriptive as the rules handcoded in rftell, or sufficient for providing a satisfactory description to all actions.

Before jumping into more complex models, one possible approach would to fill that conceptual hole: repeat the experiments above, but include the possibility for the no-change transition, fixing a specific state to a variable in the given frame.

Transition clauses

I define each particle of a RFTELL rule to be a *transition clause.* Each of those may take one of two forms: a single transition in a variable, or a single transition, accompanied by a requirement that another variable remains constant at the same value. For example, the rule for give,

```
C0,1=0,1;C0,2=1;;C0,2=1,0;C0,1=1;;
```

is made of two such clauses; the first one specifies that contact between actors 0 and 1 is made, while contact between actors 0 and 2 exists. If the rule were to be

```
C0,1=0,1;;C0,2=1,0;C0,1=1;;
```

then the first clause would have no requirement that other variables remain constant.

Extracting those clauses from event traces is similar to extracting just transitions. Now, for each frame on which various x transitions happen, there are $x(N - x + 1)$ possible transition clauses to extract, based on the combinatorics of which state variables change and which state variables are fixed in a given frame. The problem size increases at most by a constant factor of N, causing no big issues with the running time.

3.4.2 Physical state

Looking closely at supervised learning and to results from RFTELL, the main modeling issue seemed to be the inability to represent state along with transition. At the ideal state, one would be able to capture rules as the ones hardcoded in RFTELL: some variables changing, and some variables (but not all) fixed. Determining which variables should be captured as fixed, however, greatly increases the complexity of the problem.

Another issue that also could be better explored is timing: as seen on RFTELL and on various patterns found from supervised learning, very similar actions would not trigger the same pattern because actions that were only a couple frames apart would be detected in the reverse order.

In order to deal with timing, instead of using transitions, future work could investigate using sets of transitions: transitions that occur too closely together can be put on a single set, and the order in which they occur on an event trace should not be relevant. Ideally this would remove any ordering artifacts introduced in earlier stages of the system. Each transition should not appear in more than one set, and sets should be divided based on time windows, i.e., based on frame numbers. The current research would be a specific case of this model, where each set is limited to a single transition.

The trie structure could then be modified: each node now would have a state representation, and each edge will have a transition set. A pattern, read from root to leaf, would again describe the complete change of physical state on a system. Possibly, patterns with the same sequence of transition sets will then be collected, and the information for which parts of the state remain will be extracted. How to do so in unsupervised learning is also left for future work.

The trie structure would be used both for supervised and unsupervised learning, built from collected input; the only difference would be which input is used for each experiment. In the supervised case, a single pattern could be picked as the definition, again by using heuristics of least common variable types. In the unsupervised case, relevant patterns might still require some other mechanism for being selected from the results.

The target result in either case would be rules similar to the ones hardcoded in RFTELL. Detection becomes just a matter of matching; prediction, again, can be done based on prior observed patterns and weights on the trie structure.

This direction of future work, however, was abandoned in favor of a more promising approach, described in the next chapter.

Chapter 4

Boosting

This chapter details boosting classifiers: the transition-based patterns of chapter 3 are used as weak classifiers for a boosting algorithm, producing classifier ensembles that achieve perfect classification on the training data, and intuitive descriptions.

Pattern extraction, and producing a library of candidate classifiers of a manageable size, are discussed in subsection 4.1.1; the algorithm itself is discussed in subsections 4.1.2 and 4.1.3.

Results for the supervised learning for each of the 16 verbs are shown and discussed in section 4.2, and discussed in section 4.3.

When done with this chapter, the reader be familiar with the central results of my research: classifier ensembles as intuitive and good-fitting mechanisms for visuospatial action recognition.

Simple patterns turned out to be insufficient to describe an action by themselves. Intuitive patterns selected by hand (table 4.1) were not unique to the actions that they were meant to describe, and patterns that were unique, when they existed, did not appear to provide an intuitive description of the scene (see table 4.2).

While this might indicate an inadequacy of the selected representation, examining the occurrences of hand-picked patterns across events suggests that the logs produced from samples of distinct verbs are different enough from each other to make classification possible, even if the found patterns are not unique identifiers.

Table 4.1: Hand-picked pattern descriptions; see table 3.2 for a legend for transitions

name	pattern
approach	$R_{0,1} = 0, -1$; $R_{0,1} = -1, 0$;
bounce	$Y_0 = 0, -1$; $Y_0 = -1, 0$; $Y_0 = 0, 1$;
carry	$C_{0,1} = 0, 1$; $M_0 = 0, 1$;
catch	$C_{0,1} = 0, 1$; $M_0 = 1, 0$;
collide	$C_{0,1} = 0, 1$; $R_{0,1} = -1, 0$;
drop	$Y_0 = 0, -1$; $C_{0,1} = 1, 0$;
fly_over	$W_{0,1} = 0, 1$; $W_{0,1} = 1, 0$;
follow_left	$X_0 = 0, -1$; $X_1 = 0, -1$;
follow_right	$X_0 = 0, 1$; $X_1 = 0, 1$;
give	$C_{0,1} = 0, 1$; $C_{0,2} = 1, 0$;
hit	$R_{0,1} = 0, -1$; $C_{0,1} = 0, 1$;
jump	$Y_0 = 0, 1$; $Y_0 = 1, 0$; $Y_0 = 0, -1$;
pick_up	$C_{0,1} = 0, 1$; $Y_0 = 0, 1$;
push	$C_{0,1} = 0, 1$; $M_0 = 0, 1$;
put_down	$Y_0 = -1, 0$; $C_{0,1} = 1, 0$;
take	$R_{0,1} = 0, -1$; $C_{0,1} = 0, 1$; $C_{0,2} = 1, 0$;
throw	$C_{0,1} = 1, 0$; $M_0 = 0, 1$;

Observing different patterns occur with different frequencies across distinct samples hints at looking at a method that combines multiple patterns in order to build a single classifier – in particular, boosting. Results in table 4.2 motivate boosting, suggesting secondary patterns that corroborate or reinforce main patterns.

4.1 Description of boosting

Boosting is the name given to learning algorithms that create a stronger learner from a set of weak learners. In particular, the system uses AdaBoost, originally formulated by Freund and Schapire [Freund and Schapire, 1995], to improve the performance in this classification problem.

Each candidate patterns is used for two potential weak classifiers: one that returns positive for the presence of the patterns as a subsequence in the transition log and negative for its absence, and another classifier that returns the reverse result.

Any permutation of the actors is permissible during the check for a subsequence

Table 4.2: Occurrence of hand-picked patterns in given samples: a pattern is detected if its definition (table 4.1) occurs as a subsequence in the event log

sample	detected patterns
approach0	approach fly_over jump
approach1	approach follow_left
approach2	approach hit
bounce0	approach bounce fly_over jump
bounce1	approach bounce fly_over jump
bounce2	bounce jump
carry0	carry catch fly_over follow_left push
carry1	approach carry catch collide fly_over follow_left jump pick_up push
carry2	approach bounce carry catch collide fly_over follow_right hit jump pick_up push throw
catch0	approach catch collide fly_over hit jump
catch1	approach catch collide fly_over hit pick_up
catch2	approach catch collide fly_over follow_right hit
collide0	approach catch collide fly_over follow_right hit
collide1	approach carry catch collide hit push
collide2	approach bounce catch collide drop fly_over hit put_down
drop0	carry catch drop fly_over push throw
drop1	(none: bad sample file)
drop2	carry catch drop fly_over push throw
fly_over0	approach bounce fly_over follow_left jump
fly_over1	bounce fly_over follow_left jump
fly_over2	jump
follow0	approach bounce follow_left
follow1	approach fly_over follow_left
follow2	approach catch collide fly_over follow_right hit pick_up
give0	approach carry catch collide fly_over follow_right give hit push take throw
give1	approach carry catch collide fly_over follow_left give hit push take throw
give2	approach carry catch collide fly_over give hit jump pick_up push take throw
hit0	approach collide fly_over hit
hit1	approach fly_over
hit2	drop fly_over hit
jump0	bounce jump
jump1	bounce jump
jump2	approach bounce fly_over jump
pick_up0	approach carry catch collide fly_over hit pick_up push
pick_up1	approach carry catch collide fly_over hit pick_up push throw
pick_up2	approach carry catch collide fly_over hit pick_up push
push0	approach carry catch collide fly_over follow_right hit pick_up push throw
push1	approach bounce carry catch collide fly_over follow_left hit push put_down throw
push2	approach carry catch collide fly_over follow_left hit push put_down throw
put_down0	carry catch drop fly_over push put_down
put_down1	carry catch drop fly_over push put_down
put_down2	carry catch drop fly_over push put_down throw
take0	approach carry catch collide fly_over give hit pick_up push take throw
take1	approach carry catch collide fly_over give hit pick_up push take throw
take2	approach carry catch collide fly_over give hit pick_up take push
throw0	carry jump pick_up push throw
throw1	carry drop fly_over give hit jump pick_up push put_down
throw2	bounce carry catch fly_over follow_left jump pick_up push throw

Table 4.3: Library sizes obtained give samples only (see subsection 4.1.1)

	size 1	size 2	size 3	total
no F, W, M, E	8	49	200	257
no F	13	184	1982	2179
all transitions	15	261	3442	3718

Table 4.4: Full library sizes (see subsection 4.1.1)

	size 1	size 2	size 3	total
no F, W, M, E	18	545	12009	12572
no F	24	766	13762	14552
all transitions	26	906	19014	19946

match, and patterns are stored in an unique representation, as explained previously. Again, the maximum distance between consecutive matching transitions is limited to at most 60 frames.

4.1.1 Candidate classifiers

The pattern used for potential weak classifiers should be patterns obtained from training examples. The simple approach used was to simply select all patterns that occur in at least 3 samples; this provides a large library of patterns, increasing the expressivity of possible ensemble classifiers, but making the classfication process rather slow.

Variations were attempted as to which types of transitions should be considered for the library – all types, all types except focus transitions (F), and all types except focus transitions, existence, movement without direction and orientation change (F, E, M and W, respectively); information on the sizes of libraries for each of the different configurations are included in tables 4.3 and 4.4.

4.1.2 Classifier algorithm

Each classifier keeps a copy of the pattern it is based on, whether it is an inverting classifier, and its weight α. It also keeps a *pattern watcher*, a class responsible for observing each transition and notifying the classifier when the pattern has been completed; once the whole event log has been observed, the classifier can return the result, based on whether it has been notified by the pattern watcher, and whether it is an inverting classifier.

The pattern watcher keeps a list of *completion watchers*, each of which represents a potential match between the observed transition log and the pattern. Whenever a new transition is observed by a pattern watcher:

1. For each existing completion watcher, observe the new transition, and obtain the list of new completion watchers.

2. Add all new completion watchers to the list of the pattern watcher.

3. For each completion watcher: if it has completed the pattern, notify the classifier, and remove the completion watcher. If the frame number of the last observed transition was too high (more than 60 frames apart), remove the completion watcher.

4. Keep track of the largest position from a completion watcher, and produce it for displaying the progress of the classifier.

Each completion watcher keeps a reference to the pattern it is observing, a map of the actors that have already been fixed, the position of the pattern that has already been fixed, and the frame number of the last observed matching frame. Whenever it observes a new transition:

1. If the new transition cannot match the next transition in its pattern, given the fixed actors, return an empty list of new completion watchers.

2. Given the current match of actors, for each new possible match of actors, create a new completion watcher with the new match of the actors, the same pattern, and one position further, then return a list of all new completion watchers.

In effect, each completion watcher keeps track of one possible match, each pattern classifier keeps track of the all possible matches for a pattern, and the weak classifier itself is responsible for the transition log classification.

Because transition logs are immutable, the presence of a pattern as a subsequence only needs to be checked once, and can then be memoized for later use.

4.1.3 Boosting algorithm

In order to use a classifier ensemble, each weak classifier that is part of the hypothesis contributes with its classification c_i, which can be 1 or -1, and its weight α_i. If $\sum c_i\alpha_i > 0$, the ensemble classification is 1, otherwise it is -1.

The pseudo-code for a step of the boosting algorithm is as follows:

1. Pick the best unused classifier c from the set of candidate classifiers C; if there are no unused classifiers left, stop. All unused classifiers start with the same weight.

2. If c is an inverting classifier, invert it to a positive classifier, and change the sign of its weight.

3. If there is a matching classifier in the current hypothesis set, increase its weight by the weight of c; otherwise, add c to the current hypothesis set.

4. Let e be the number of samples classified incorrectly by c, and $|S|$ the total number of samples. $\epsilon = e/|S|$ is the error of the classifier c. For each sample s, if the current ensemble of classifiers classifies s correctly, multiply the weight of s by $0.5/(1-\epsilon)$, else multiply it by $0.5/\epsilon$.

5. Normalize all sample weights.

Steps are taken until there are no classification errors left, or until there are no candidate classifiers left (or until the algorithm is manually interrupted).

4.2 Boosting results

Preliminary results were less than promising: before running the experiment with the full pattern library as a list of candidate weak classifiers, boosting was tested on a smaller number of examples, using a smaller number of classifiers.

At first, limiting transitions to not include F, E, T or W, only patterns that appeared in all *give* training examples were used. The resulting list of patterns was insufficient to build a classifier that correctly separated all gives from the other patterns, or all approaches from the other patterns.

As a second test, the hand-picked patterns for each of the 16 verbs were used as candidates for classifiers; again, it was impossible to build an ensemble able to properly classify all training data.

However, once the tests were ran with the full library of patterns, using all possible transitions, all actions were perfectly classified – not a surprise, because some of the verbs did have non-intuitive unique patterns. A sample classifier ensemble is shown in the following sections.

Ensemble pictures

The pictures in the Appendix A are screen shots capture of the final state of the classifier program. In the main panel, each line represents a training sample: the first column is the sample name, the second column marks whether it was labeled as a *hit* or as a *miss*, the third column shows whether its classification by the current classifier is correct, and the fourth column contains the sequence of transitions composing the sample itself.

In the right panel, the list of selected classifiers so far shows up, listing weight and the pattern used in its canonical form (all actors permutated to give the first possible lexicographical representation of the pattern).

Table 4.5: Approach classifier ensemble

alpha	pattern	description
1.925	$X_0 = -1, 0$; $S_0 = 1, 0$;$X_0 = 0, -1$;	actor 0 stops moving left actor 0 stops increasing in size actor 0 starts moving left
1.557	$R_{0,1} = 1, 0$; $X_0 = 0, -1$; $C_{0,1} = 0, 1$;	actors 0 and 1 stop moving apart actor 0 starts moving left actors 0 and 1 make contact
2.697	$R_{0,1} = 1, 0$; $X_0 = -1, 0$; $C_{0,1} = 1, 0$;	actors 0 and 1 stop moving apart actor 0 stops moving left actors 0 and 1 break contact
-2.375	$R_{0,1} = 0, -1$; $R_{0,1} = 0, 1$;	actors 0 and 1 start approaching each other actors 0 and 1 start moving apart
-2.292	$C_{0,1} = 0, 1$; $R_{0,1} = 0, 1$;	actors 0 and 1 make contact actors 0 and 1 start moving apart

In the bottom panel, a graph shows the number of classification errors at each boosting iteration step.

4.2.1 Approach

The resulting ensemble for classifying the approach samples is shown in figure A-1, and described at table 4.5. The desired pattern for approach appears as a subpattern of many other samples that were here selected as misses instead of hits. As a result, the boosting algorithm selected for patterns that are specific to the given samples of approach, selecting transitions that do not seem too related to the action of approaching itself – such as changes of size, movement of one of the actors, and contact: overfitting, given the small data set. It also selected against actors first approaching each other and then moving farther apart again (fourth pattern).

4.2.2 Bounce

The resulting ensemble for classifying bounce samples is shown in figure A-2, and described at table 4.6. All the selected patterns relate to the vertical movement of a

Table 4.6: Bounce classifier ensemble

alpha	pattern	description
1.925	$Y_0 = 0, -1$; $Y_0 = 0, -1$; $Y_0 = 1, 0$;	actor 0 starts moving down actor 0 starts moving down actor 0 stops moving up
2.267	$R_{0,1} = 0, -1$; $Y_0 = -1, 0$;$Y_0 = -1, 0$;	actors 0 and 1 start approaching each other actor 0 stops moving down actor 0 stops moving down
2.260	$Y_0 = 0, -1$; $Y_0 = 1, 0$; $Y_0 = -1, 0$;	actor 0 starts moving down actor 0 stops moving up actor 0 stops moving down

Table 4.7: Carry classifier ensemble

alpha	pattern	description
1.568	$X_0 = 1, 0$; $X_0 = 0, 1$; $X_1 = 1, 0$;	actor 0 stops moving right actor 0 starts moving right actor 1 stops moving right
1.695	$C_{0,1} = 0, 1$; $X_0 = 0, -1$; $X_1 = 0, -1$;	actors 0 and 1 make contact actor 0 starts moving left actor 1 stops moving left
-0.979	$C_{0,1} = 0, 1$; $C_{0,1} = 1, 0$;	actors 0 and 1 make contact actors 0 and 1 break contact

single actor, as the desired pattern, but they often skip some of the vertical transitions in the event log. The ensemble is similar, but not the same, as the desired pattern, and successfully classifies the jump examples as misses, unlike the desired pattern.

4.2.3 Carry

As seen on figure A-3 and table 4.7, the first pattern specifies only movement, and no contact: actor 0 comes from the left, stops, and gets actor 1, moving it along with it right, until it stops. On the second pattern, contact is noticed, as well as both actors starting to move. The last negative pattern removes the cases where contact is made and then later broken – in all testing samples, the carrying actor never lets go of the ball, unlike in some other actions. Those examples are similar to the desired pattern,

Table 4.8: Catch classifier ensemble

alpha	pattern	description
1.925	$Y_0 = 0, -1$; $C_{0,1} = 0, 1$;$Y_0 = -1, 0$;	actor 0 starts moving down actors 0 and 1 make contact actor 0 stops moving down
1.706	$C_{0,1} = 0, 1$; $Y_0 = -1, 0$; $R_{0,1} = -1, 0$;	actors 0 and 1 make contact actor 0 stops moving down actors 0 and 1 stop approaching
3.097	$R_{0,1} = 0, -1$; $Y_0 = -1, 0$; $X_0 = -1, 0$;	actors 0 and 1 start approaching actor 0 stops moving down actor 0 stops moving left

Table 4.9: Collide classifier ensemble

alpha	pattern	description
1.568	$S_0 = -1, 0$; $Y_0 = 1, 0$; $X_1 = 1, 0$;	actor 0 stops reducing in size actor 0 stops moving up actor 0 stops moving right
1.052	$R_{0,1} = -1, 0$; $R_{0,1} = 1, 0$; $C_{0,1} = 1, 0$;	actors 0 and 1 stop approaching actors 0 and 1 stop moving apart actors 0 and 1 break contact
1.812	$R_{0,1} = -1, 0$; $R_{0,1} = 0, 1$; $C_{0,1} = 1, 0$;	actors 0 and 1 stop approaching actors 0 and 1 start moving apart actors 0 and 1 break contact
-1.632	$C_{0,1} = 0, 1$; $R_{0,1} = 0, -1$;	actors 0 and 1 make contact actors 0 and 1 start approaching

and the ensemble successfully classifies only the carry samples as hits.

4.2.4 Catch

The classifier ensemble for catch can be seen at table 4.8 and figure A-4. All patterns are positive and hint at the directions the objects from which objects approach each other before the contact, or how the movement stops after contact.

Table 4.10: Drop classifier ensemble

alpha	pattern	description
1.925	$Y_0 = 0, -1$; $C_{0,1} = 1, 0$; $R_0 = 1, 0$;	actor 0 starts moving down actors 0 and 1 break contact actors 0 and 1 stop moving apart
-0.915	$R_{0,1} = -1, 0$;	actors 0 and 1 stop approaching
-1.612	$R_{0,1} = 0, -1$; $R_{0,1} = -1, 0$;	actors 0 and 1 start approaching actors 0 and 1 stop approaching
-1.313	$Y_0 = 0, -1$;	actor 0 starts moving down
0.956	$R_{0,1} = 0, -1$; $R_{0,1} = 0, 1$;	actors 0 and 1 start approaching actors 0 and 1 start moving apart
-1.603	$X_0 = 0, -1$;	actor 0 starts moving left

4.2.5 Collide

The ensemble for collide is described at table 4.9 and figure A-5. The first pattern describes only one of the actors reacting to the collision: stopping movement farther away from the camera, up and to the right. The second and third patterns describe parts of the following sequence of events: the actors approach, stop approaching, make contact, start moving away, and break contact. The last pattern, negative, indicates that once contact is made, the actors do not get any closer later.

4.2.6 Drop

As seen by the size of the ensemble in table 4.10 and figure A-6, the concept described here is much more complex. The first pattern is intuitive enough to understand, and even envelops the hand-picked pattern for drop: an object starts moving down, loses contact with the other object, and then stops moving away (due to having hitting the floor). The third pattern, negative, implies that objects should not start approaching each other and stop doing so; the second pattern reinforces the end of that. The fourth pattern, strangely enough, attributes a negative weight to an object simply moving down. The fifth pattern asks for objects first approaching and then moving further apart (a consequence of bringing an object closer before dropping it), and the

Table 4.11: Fly over classifier ensemble

alpha	pattern	description
Infinity	$X_0 = 0, 1$; $Y_1 = 0, 1$; $Y_1 = 0, -1$;	actor 0 starts moving right actor 1 starts moving up actor 1 starts moving down

Table 4.12: Follow classifier ensemble

alpha	pattern	description
1.925	$X_0 = 0, -1$; $R_{0,1} = 1, 0$; $X_0 = 0, -1$;	actor 0 starts moving left actors 0 and 1 stop moving apart actor 0 starts moving left
2.266	$R_{0,1} = 0, -1$; $S_0 = -1, 0$; $R_{0,1} = -1, 0$;	actors 0 and 1 start approaching actor 0 stops decreasing actors 0 and 1 stop approaching
2.261	$R_{0,1} = 1, 0$; $R_{0,1} = 0, -1$; $Y_0 = 1, 0$;	actors 0 and 1 stop moving apart actors 0 and 1 start approaching actor 0 stops moving up

last one gives a negative weight to any movement to the left (drops always happened straight down or slightly to the right).

4.2.7 Fly over

As seen on table 4.11 and figure A-7, there is a single pattern that classifies the samples of fly over perfectly. It appears that on fly over examples, and only on fly over examples, there is an actor moving right, while the other actor first moves up and then down. This isn't an intuitive result, but it did detect that on every scene of a ball flying over a person, the person moved right, while the ball moved up and down.

4.2.8 Follow

The description for the follow ensemble is at table 4.12 and figure A-8. The first pattern is very similar to the hand-picked concept of follow_left, but it also detects

Table 4.13: Give classifier ensemble

alpha	pattern	description
Infinity	$R_{0,1} = 1, 0$; $R_{0,2} = 0, -1$; $R_{0,1} = 0, 1$;	actors 0 and 1 start moving apart actors 0 and 2 start approaching actors 0 and 1 start moving apart

that the distance stops increasing because the leader started moving first, before the follower also started moving. The second pattern is indicative of something happening at the end of the follow action: the leader stops, the distance starts decreasing, and then it stops, as one of the actors moves away from the camera while stopping. The third pattern also detects a variation in change from the start and the end of the action, but with a vertical movement – caused by the first step of one of the actors.

4.2.9 Give

The ensemble for give, at table 4.13 and figure A-9, indicates that a single pattern can classify perfectly all the training data. It makes no mention of contact between ball and receiver: despite it apparently being a fundamental characteristic of give, the distinguishing feature in all training examples is the sequence in which objects approach or depart from each other. Better miss examples would help filter out this classification. Nevertheless, it hints that the relative movements are important to identify an action.

4.2.10 Hit

The ensemble for hit is shown at table 4.14 and A-10. The first pattern shows the movement of the moving object right after the hit – start moving away and lose contact. The second pattern shows the object approaching, stopping, and moving away again. The third pattern is a disincentive against objects moving up. Patterns 4 and 5 weight contact negatively. Patterns 6 and 7 hint at movement stopping right before contact; patterns 8 and 9 show the final fall right before contact, and then

Table 4.14: Hit classifier ensemble

alpha	pattern	description
1.568	$Y_0 = 0, -1$; $X_0 = 0, 1$; $C_{0,1} = 1, 0$;	actor 0 starts moving down actor 0 starts moving right actors 0 and 1 break contact
0.949	$R_{0,1} = 0, -1$; $X_0 = -1, 0$; $R_{0,1} = 0, 1$;	actors 0 and 1 start approaching actor 0 stops moving left actors 0 and 1 start moving apart
-1.632	$Y_0 = 0, 1$;	actors 1 starts moving up
-1.509	$C_{0,1} = 1, 0$;	actors 0 and 1 break contact
-0.931	$C_{0,1} = 0, 1$; $C_{0,1} = 1, 0$;	actors 0 and 1 make contact actors 0 and 1 break contact
1.075	$X_0 = -1, 0$; $R_{0,1} = -1, 0$; $R_{0,1} = 0, 1$;	actor 0 stops moving left actors 0 and 1 stop approaching actors 0 and 1 start moving apart
1.512	$R_{0,1} = 0, -1$; $X_0 = -1, 0$; $R_{0,1} = -1, 0$;	actors 0 and 1 start approaching actors 0 stops moving left actors 0 and 1 stop approaching
1.727	$Y_0 = 0, -1$; $C_{0,1} = 0, 1$; $R_{0,1} = 0, 1$;	actor 0 starts moving down actors 0 and 1 make contact actors 0 and 1 start moving apart
2.771	$Y_0 = 0, -1$; $C_{0,1} = 0, 1$; $X_0 = -1, 0$;	actor 0 starts moving down actors 0 and 1 make contact actor 0 stops moving left

Table 4.15: Jump classifier ensemble

alpha	pattern	description
1.354	$R_{0,1} = 0,1;\ X_2 = 0,-1;\ Y_2 = -1,0;$	actors 0 and 1 start moving apart actor 2 starts moving left actor 2 stops moving up
1.534	$Y_0 = -1,0;\ Y_0 = 0,-1;\ Y_0 = 0,1;$	actor 0 stops moving down actor 0 starts moving down actor 0 starts moving up
-0.808	$R_{0,1} = 0,1;$	actors 0 and 1 start moving apart
0.779	$S_0 = -1,0;\ S_0 = 1,0;\ R_{0,1} = 0,1;$	actor 0 stops decreasing in size actor 0 stops increasing in size actors 0 and 1 start moving apart
1.311	$R_{0,1} = -1,0;\ S_0 = 1,0;\ X_1 = -1,0;$	actors 0 and 1 stop approaching actor 0 stops increasing in size actor 1 stops moving left
-1.468	$Y_0 = 0,-1;\ Y_0 = 0,-1;\ Y_0 = 0,1;$	actor 0 starts moving down actor 0 starts moving down actor 1 starts moving up

distance increasing after it (8) or a horizontal stop (9).

4.2.11 Jump

The ensemble for jump can be seen at table 4.15 and figure A-11. Most samples of jump include more than one actor in this training set; patterns are detected that reflect that. Nevertheless, a strong positive pattern is detected, indicating vertical movement, and a strong negative one is also in place to invalidate very similar bounce samples (with the two downward movements).

4.2.12 Pick up

The obtained ensemble for pick up is at table 4.16 and figure A-12. Patterns 1 and 4 hint at contact happening and being broken multiple times – a consequence of the filtering and the very tenuous contact between carried object and carrier. Patterns 2 and 3 display the relative movements of the carrier and carried objects, but fail

Table 4.16: Pick up classifier ensemble

alpha	pattern	description
1.925	$C_{0,1} = 1, 0$; $R_{0,1} = -1, 0$; $C_{0,1} = 0, 1$;	actors 0 and 1 break contact actors 0 and 1 stop approaching actors 0 and 1 make contact
1.914	$X_0 = 0, -1$; $X_1 = 1, 0$; $Y_1 = 1, 0$;	actor 0 starts moving left actor 1 stops moving right actor 1 stops moving up
3.503	$X_0 = 0, -1$; $X_1 = 0, 1$; $Y_1 = 1, 0$;	actor 0 starts moving left actor 1 starts moving right actor 1 stops moving up
3.849	$C_{0,1} = 0, 1$; $C_{0,1} = 0, 1$; $C_{0,1} = 0, 1$;	actors 0 and 1 make contact actors 0 and 1 make contact actors 0 and 1 make contact

to include contact information. The combination of those, however, classifies the samples correctly.

4.2.13 Push

The ensemble for push can be viewed at table 4.17 and figure A-13. The first pattern shows the movement change before contact is made; the second shows movement change once contact is lost, and the third pattern shows contact being lost, made, and lost again potentially a filtering issue.

4.2.14 Put down

The ensemble for put down, at table 4.18 and figure A-14, is made of a single pattern, that classifies perfectly all training samples. It is a very intuitive one as well: the distance stops increasing as the object is being moved down, the object stops moving down, and then the contact between the object and the actor is lost.

Tablc 4.17: Push classificr cnscmblc

alpha	pattern	description
1.925	$Y_0 = -1, 0$; $X_0 = 0, -1$; $C_{0,1} = 0, 1$;	actor 0 stops moving down actor 0 starts moving left actors 0 and 1 make contact
2.266	$C_{0,1} = 1, 0$; $X_0 = 0, 1$; $Y_0 = 0, 1$;	actors 0 and 1 break contact actor 0 starts moving right actor 0 starts moving up
1.701	$C_{0,1} = 1, 0$; $C_{0,1} = 0, 1$; $C_{0,1} = 1, 0$;	actors 0 and 1 break contact actors 0 and 1 make contact actors 0 and 1 break contact

Table 4.18: Put down classifier ensemble

alpha	pattcrn	dcscription
Infinity	$R_{0,1} = 1, 0$; $Y_0 = -1, 0$; $C_{0,1} = 1, 0$;	actors 0 and 1 stop moving apart actor 0 stops moving down actors 0 and 1 losc contact

Table 4.19: Take classifier ensemble

alpha	pattcrn	dcscription
Infinity	$C_{0,1} = 0, 1$; $R_{0,2} = -1, 0$; $R_{1,2} = 1, 0$;	actors 0 and 1 make contact actors 0 and 2 stop approaching actors 1 and 2 stop moving apart

Table 4.20: Throw classifier ensemble

alpha	pattern	description
1.925	$Y_0 = 0, 1$; $C_{0,1} = 1, 0$; $Y_0 = 0, -1$;	actor 0 starts moving up actors 0 and 1 break contact actor 0 starts moving down
1.914	$Y_0 = 0, -1$; $Y_0 = 0, 1$; $X_0 = 0, 1$;	actor 0 starts moving down actors 0 starts moving up actor 0 starts moving right
3.503	$C_{0,1} = 0, 1$; $Y_0 = 0, 1$; $X_0 = 0, -1$;	actors 0 and 1 make contact actors 0 starts moving up actor 0 starts moving left
3.503	$Y_0 = 0, 1$; $R_{0,1} = 0, 1$; $Y_0 = 1, 0$;	actor 0 starts moving up actors 0 and 1 start moving apart actor 0 stops moving up

4.2.15 Take

The ensemble for take (table 4.19, figure A-15) is also made of a single pattern. As in give, not all contact information is registered by it, but information about varying distances carry the necessary information to distinguish between take and give, and all other training examples: namely, the taker approaches the giver and the ball to get it, instead of having the giver approach.

4.2.16 Throw

The ensemble for throw (table 4.20, figure A-16) has all patterns telling parts of the same story: the object is picked up moved up, loses contact, stops moving up, moves down, and bounces.

4.3 Summary and discussion

Before performing the experiments, I had two main concerns about ways it could fail: overfitting the training data, and insufficient candidate classifiers. Here, the set of potential classifiers is finite and pre-determined based on the training set, because the full space of possible patterns is intractably large. The expressivity of potential ensemble classifiers depends on how well those patterns divide the data.

The first attempt at generating a library with patterns of even size 4, from all events, required more memory than available. I scaled the libraries back to patterns of size at most 3, which were much more manageable.

In initial experiments, using only the library obtained by getting patterns common to all *give* training samples, there was insufficient information to classify any of the verbs perfectly, without false positives or negatives. The candidates present only in *give* were not expressive enough, even if the training samples for give produce some of the most verbose event logs, second only *take*. Given this, I gave up on building classifiers based on other specific verbs, and focused on the classifiers using the full library.

Eliminating variables that suggested less information and seemed less relevant to actions themselves also reduced the size of libraries, as shown in tables 4.3 and 4.4. The experiments were still far from real time, with duration in the order of minutes.

The first experiment with the full library, using the smallest of those, produced perfect classification for each of the verbs, as analyzed above. The first experiment already showed that the library it used contained enough diversity in its patterns to be able to classify all actions, as expected.

Past the issue of there not being enough classifiers, by choice of an adequate library, the next issue to be concerned about was overfitting. Some of the verbs were, in fact, defined perfectly by a single pattern; as seen on each of those cases, however, that pattern does occur in each of the training videos of the selected verb in vsr1, and nowhere else; and even those single patterns make intuitive sense, as described above.

The main result of my research is the perfect, and intuitive, classification given above. In future work, this approach should be attempted on larger training sets, to increase confidence in the results.

Chapter 5

Results and Contributions

5.1 Results

Some goals of my research were accomplished: the visuospatial action recognition system, as presented in this research, is able to identify the 16 actions perfectly within the training set, providing intuitive high-level descriptions. Recognition can occur in real time, but learning does not. The system still should be tested with larger datasets.

The results also validate the use of a transition-based representation for action, though it is likely possible to improve greatly upon the representation taken for my research, in particular the representation of trajectories, and what actors initiate actions involving another actor.

5.2 Contributions

In this section, I list the noteworthy contributions in this thesis.

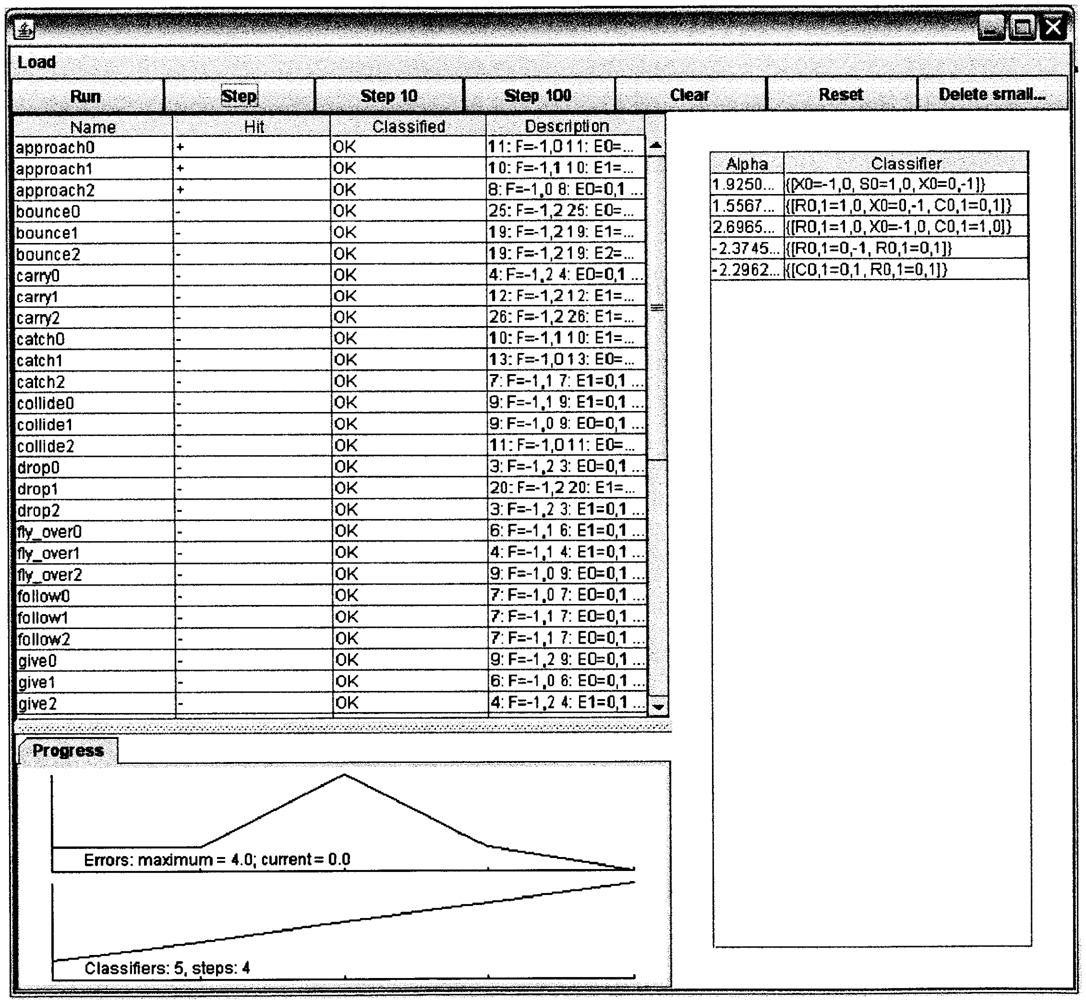
- Analyzed possible state-based transition representations for actions, and implemented one that obtained results;
- Studied possible implementations of classifiers as simple finite state machines for transitions, and concluded this to be an unsatisfactory mechanism;

- Performed several experiments with patterns as classifiers that yielded unsatisfactory results, suggesting directions for improvement and future research;
- Demonstrated that intuitive classifier ensembles can be constructed, using commonly observed patterns and boosting, to classify perfectly a given set of videos.
- Created a system that perfectly recognized all videos for each of the 16 actions: approach, bounce, carry, catch, collide, drop, fly over, follow, give, hit, jump, pick, push, put, take, throw.

Appendix A

Ensemble pictures

This appendix includes screenshoots showing the final state of the boosting classifier described in chapter 4. For each experiment, vsr1 samples corresponding to the target verb were used as hit samples, while all other vsr1 samples were used as miss samples. An explanation of the interface and a detailed discussion of each result can be found in section 4.2.



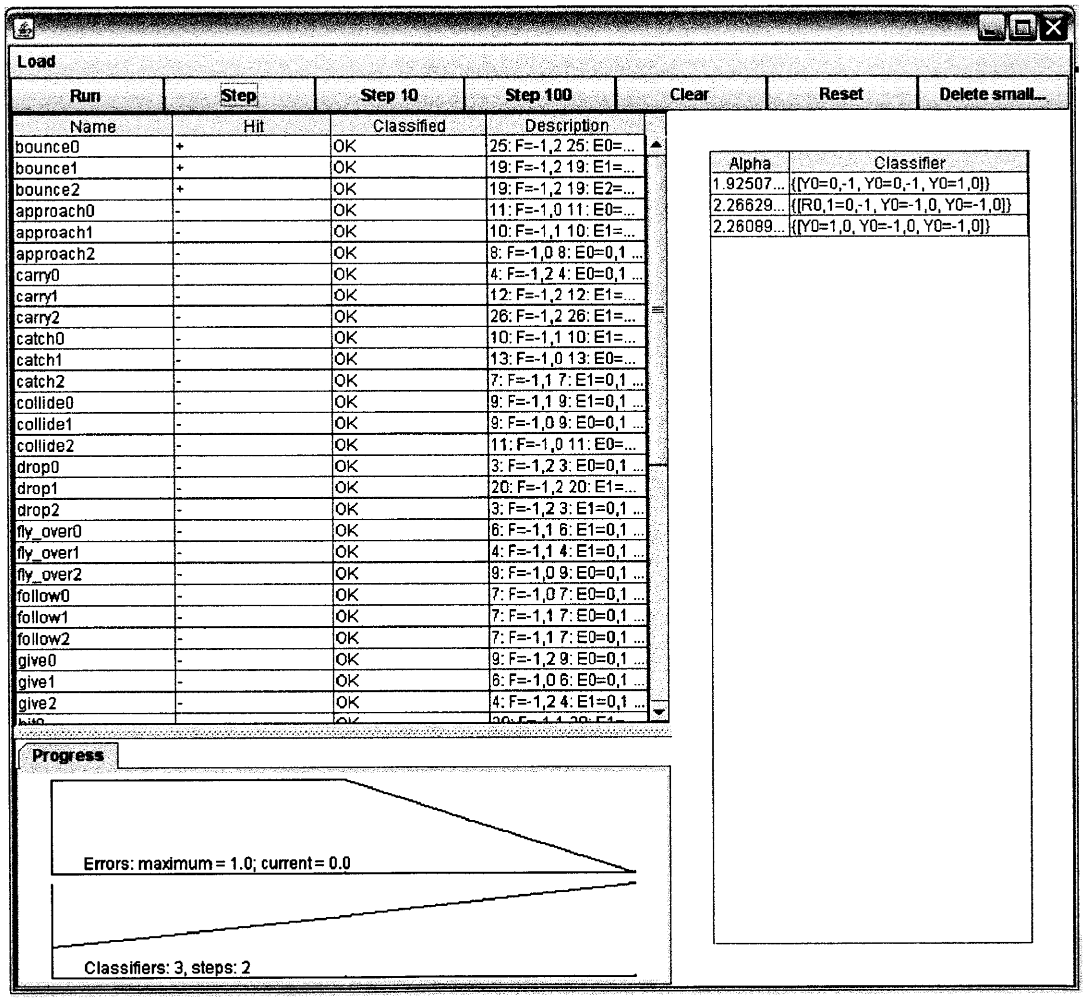
Name	Hit	Classified	Description
approach0	+	OK	11: F=-1,0 11: E0=...
approach1	+	OK	10: F=-1,1 10: E1=...
approach2	+	OK	8: F=-1,0 8: E0=0,1 ...
bounce0	-	OK	25: F=-1,2 25: E0=...
bounce1	-	OK	19: F=-1,2 19: E1=...
bounce2	-	OK	19: F=-1,2 19: E2=...
carry0	-	OK	4: F=-1,2 4: E0=0,1 ...
carry1	-	OK	12: F=-1,2 12: E1=...
carry2	-	OK	26: F=-1,2 26: E1=...
catch0	-	OK	10: F=-1,1 10: E1=...
catch1	-	OK	13: F=-1,0 13: E0=...
catch2	-	OK	7: F=-1,1 7: E1=0,1 ...
collide0	-	OK	9: F=-1,1 9: E1=0,1 ...
collide1	-	OK	9: F=-1,0 9: E0=0,1 ...
collide2	-	OK	11: F=-1,0 11: E0=...
drop0	-	OK	3: F=-1,2 3: E0=0,1 ...
drop1	-	OK	20: F=-1,2 20: E1=...
drop2	-	OK	3: F=-1,2 3: E1=0,1 ...
fly_over0	-	OK	6: F=-1,1 6: E1=0,1 ...
fly_over1	-	OK	4: F=-1,1 4: E1=0,1 ...
fly_over2	-	OK	9: F=-1,0 9: E0=0,1 ...
follow0	-	OK	7: F=-1,0 7: E0=0,1 ...
follow1	-	OK	7: F=-1,1 7: E0=0,1 ...
follow2	-	OK	7: F=-1,1 7: E0=0,1 ...
give0	-	OK	9: F=-1,2 9: E0=0,1 ...
give1	-	OK	6: F=-1,0 6: E0=0,1 ...
give2	-	OK	4: F=-1,2 4: E1=0,1 ...

Alpha	Classifier
1.9250...	{[X0=-1,0, S0=1,0, X0=0,-1]}
1.5567...	{[R0,1=1,0, X0=0,-1, C0,1=0,1]}
2.6965...	{[R0,1=1,0, X0=-1,0, C0,1=1,0]}
-2.3745...	{[R0,1=0,-1, R0,1=0,1]}
-2.2962...	{[C0,1=0,1, R0,1=0,1]}

Errors: maximum = 4.0; current = 0.0

Classifiers: 5, steps: 4

Figure A-1: Classifier ensemble for approach



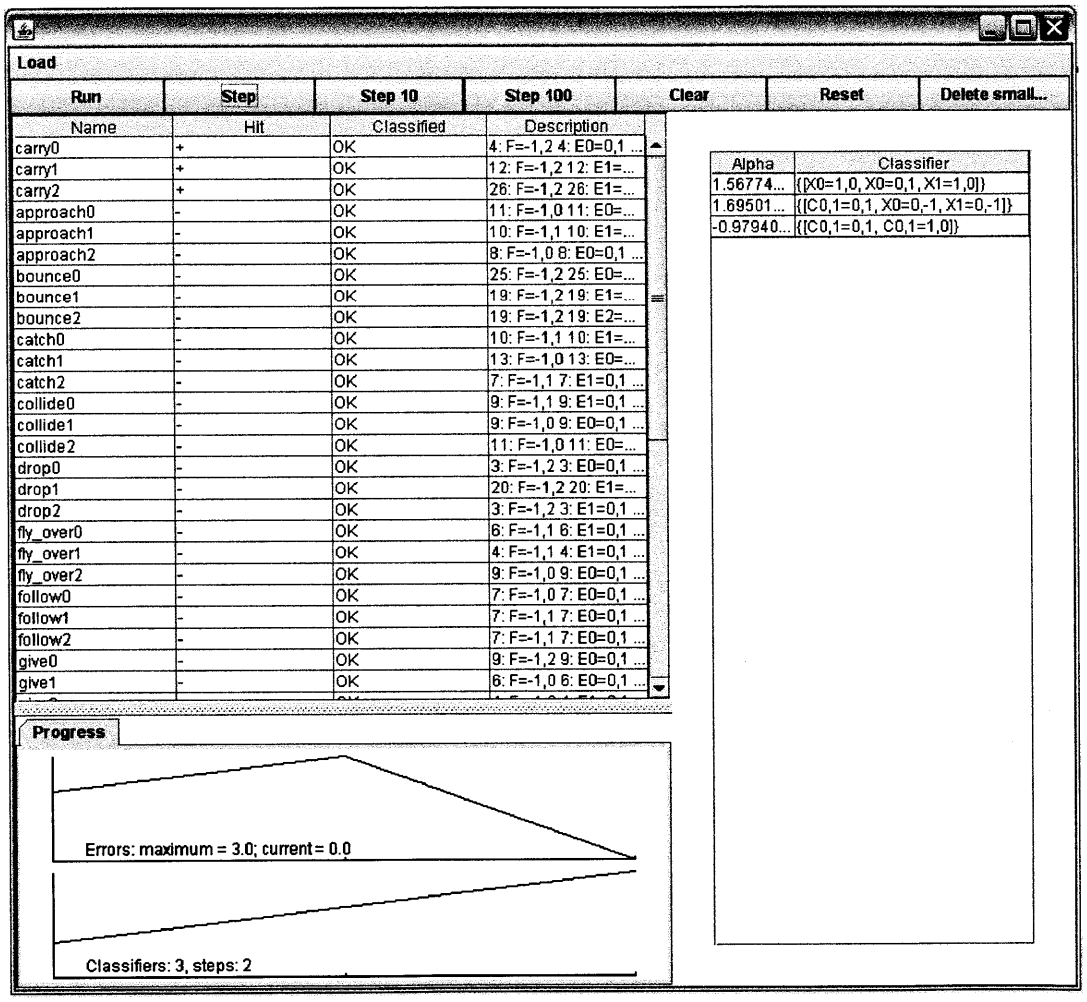
Name	Hit	Classified	Description
bounce0	+	OK	25: F=-1,2 25: E0=...
bounce1	+	OK	19: F=-1,2 19: E1=...
bounce2	+	OK	19: F=-1,2 19: E2=...
approach0	-	OK	11: F=-1,0 11: E0=...
approach1	-	OK	10: F=-1,1 10: E1=...
approach2	-	OK	8: F=-1,0 8: E0=0,1 ...
carry0	-	OK	4: F=-1,2 4: E0=0,1 ...
carry1	-	OK	12: F=-1,2 12: E1=...
carry2	-	OK	26: F=-1,2 26: E1=...
catch0	-	OK	10: F=-1,1 10: E1=...
catch1	-	OK	13: F=-1,0 13: E0=...
catch2	-	OK	7: F=-1,1 7: E1=0,1 ...
collide0	-	OK	9: F=-1,1 9: E1=0,1 ...
collide1	-	OK	9: F=-1,0 9: E0=0,1 ...
collide2	-	OK	11: F=-1,0 11: E0=...
drop0	-	OK	3: F=-1,2 3: E0=0,1 ...
drop1	-	OK	20: F=-1,2 20: E1=...
drop2	-	OK	3: F=-1,2 3: E1=0,1 ...
fly_over0	-	OK	6: F=-1,1 6: E1=0,1 ...
fly_over1	-	OK	4: F=-1,1 4: E1=0,1 ...
fly_over2	-	OK	9: F=-1,0 9: E0=0,1 ...
follow0	-	OK	7: F=-1,0 7: E0=0,1 ...
follow1	-	OK	7: F=-1,1 7: E0=0,1 ...
follow2	-	OK	7: F=-1,1 7: E0=0,1 ...
give0	-	OK	9: F=-1,2 9: E0=0,1 ...
give1	-	OK	6: F=-1,0 6: E0=0,1 ...
give2	-	OK	4: F=-1,2 4: E1=0,1 ...

Alpha	Classifier
1.92507...	{[Y0=0,-1, Y0=0,-1, Y0=1,0]}
2.26629...	{[R0,1=0,-1, Y0=-1,0, Y0=-1,0]}
2.26089...	{[Y0=1,0, Y0=-1,0, Y0=-1,0]}

Errors: maximum = 1.0; current = 0.0

Classifiers: 3, steps: 2

Figure A-2: Classifier ensemble for bounce



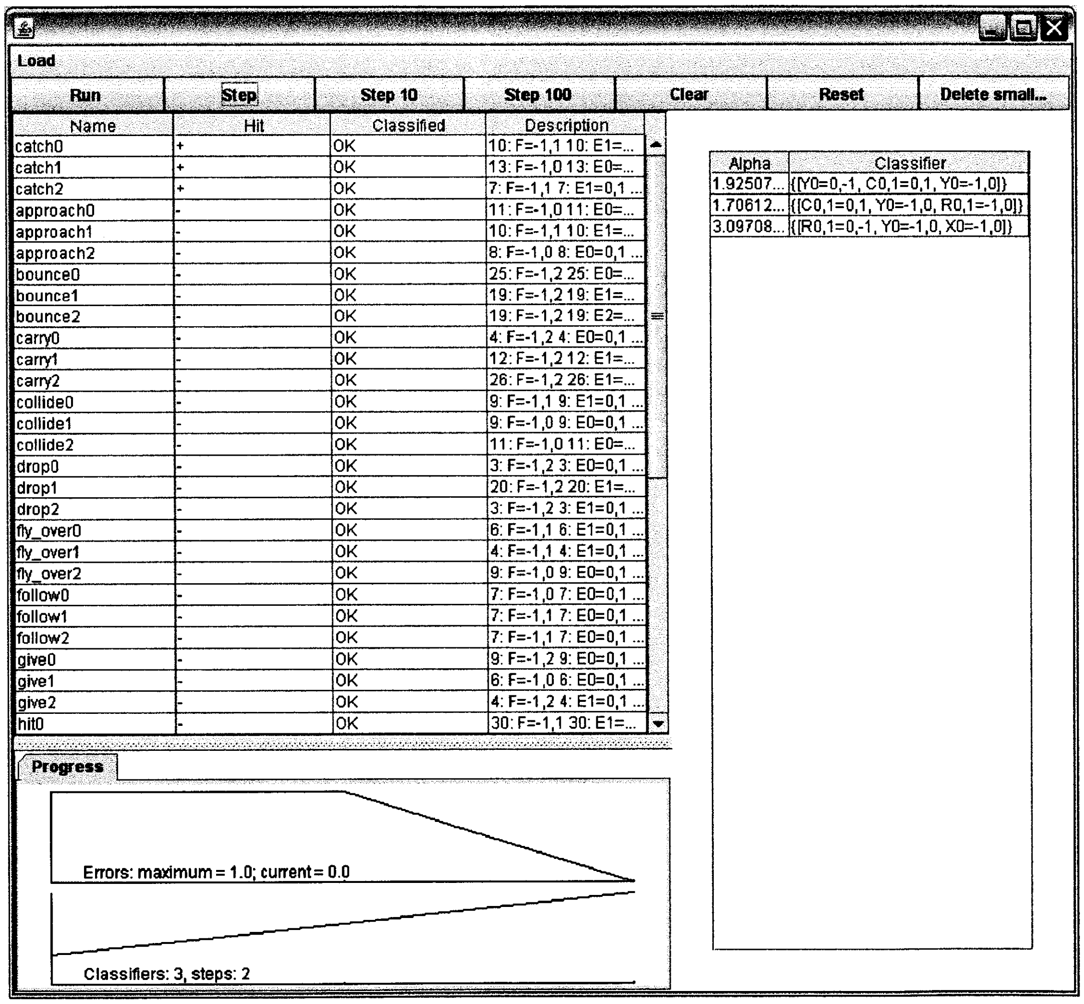
Name	Hit	Classified	Description
carry0	+	OK	4: F=-1,2 4: E0=0,1 ...
carry1	+	OK	12: F=-1,2 12: E1=...
carry2	+	OK	26: F=-1,2 26: E1=...
approach0	-	OK	11: F=-1,0 11: E0=...
approach1	-	OK	10: F=-1,1 10: E1=...
approach2	-	OK	8: F=-1,0 8: E0=0,1 ...
bounce0	-	OK	25: F=-1,2 25: E0=...
bounce1	-	OK	19: F=-1,2 19: E1=...
bounce2	-	OK	19: F=-1,2 19: E2=...
catch0	-	OK	10: F=-1,1 10: E1=...
catch1	-	OK	13: F=-1,0 13: E0=...
catch2	-	OK	7: F=-1,1 7: E1=0,1 ...
collide0	-	OK	9: F=-1,1 9: E1=0,1 ...
collide1	-	OK	9: F=-1,0 9: E0=0,1 ...
collide2	-	OK	11: F=-1,0 11: E0=...
drop0	-	OK	3: F=-1,2 3: E0=0,1 ...
drop1	-	OK	20: F=-1,2 20: E1=...
drop2	-	OK	3: F=-1,2 3: E1=0,1 ...
fly_over0	-	OK	6: F=-1,1 6: E1=0,1 ...
fly_over1	-	OK	4: F=-1,1 4: E1=0,1 ...
fly_over2	-	OK	9: F=-1,0 9: E0=0,1 ...
follow0	-	OK	7: F=-1,0 7: E0=0,1 ...
follow1	-	OK	7: F=-1,1 7: E0=0,1 ...
follow2	-	OK	7: F=-1,1 7: E0=0,1 ...
give0	-	OK	9: F=-1,2 9: E0=0,1 ...
give1	-	OK	6: F=-1,0 6: E0=0,1 ...

Alpha	Classifier
1.56774...	{[X0=1,0, X0=0,1, X1=1,0]}
1.69501...	{[C0,1=0,1, X0=0,-1, X1=0,-1]}
-0.97940...	{[C0,1=0,1, C0,1=1,0]}

Errors: maximum = 3.0; current= 0.0

Classifiers: 3, steps: 2

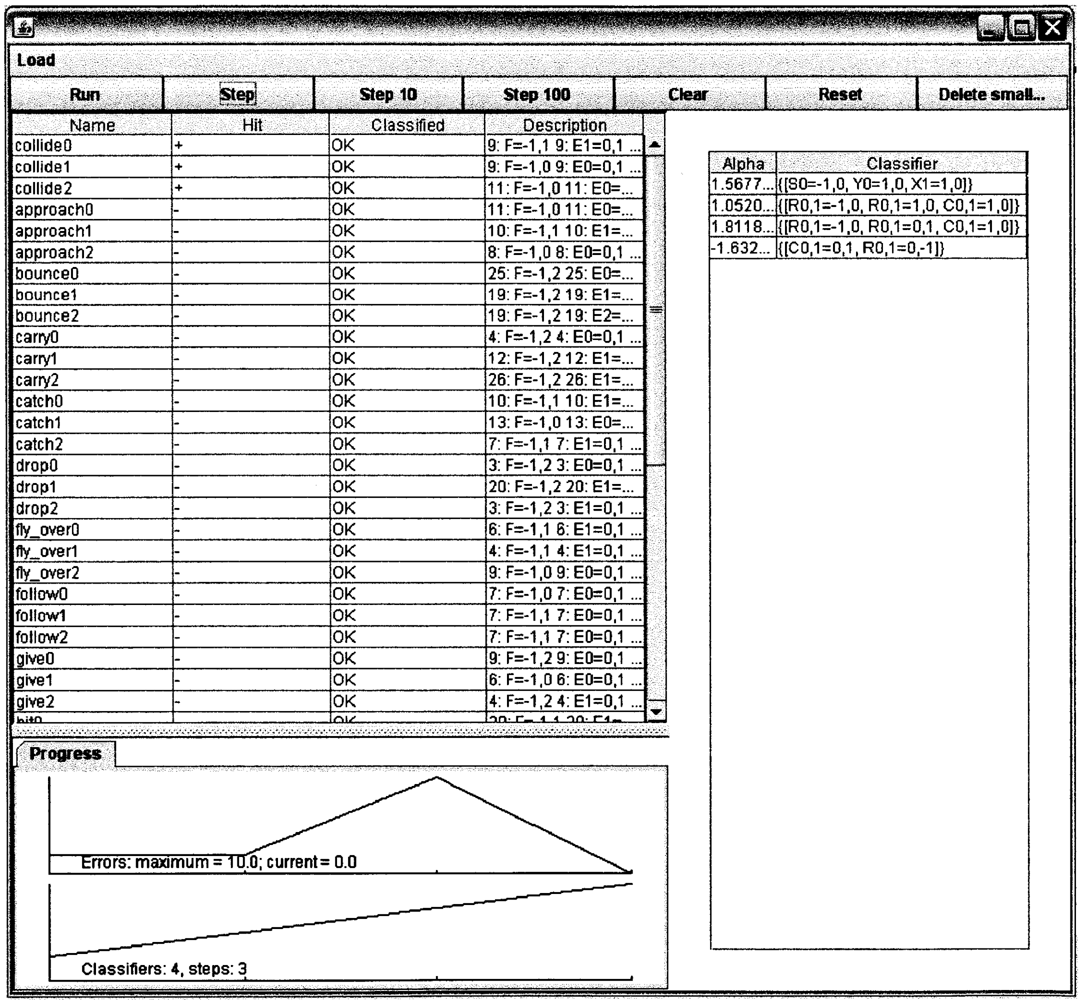
Figure A-3: Classifier ensemble for carry



Name	Hit	Classified	Description
catch0	+	OK	10: F=-1,1 10: E1=...
catch1	+	OK	13: F=-1,0 13: E0=...
catch2	+	OK	7: F=-1,1 7: E1=0,1 ...
approach0	-	OK	11: F=-1,0 11: E0=...
approach1	-	OK	10: F=-1,1 10: E1=...
approach2	-	OK	8: F=-1,0 8: E0=0,1 ...
bounce0	-	OK	25: F=-1,2 25: E0=...
bounce1	-	OK	19: F=-1,2 19: E1=...
bounce2	-	OK	19: F=-1,2 19: E2=...
carry0	-	OK	4: F=-1,2 4: E0=0,1 ...
carry1	-	OK	12: F=-1,2 12: E1=...
carry2	-	OK	26: F=-1,2 26: E1=...
collide0	-	OK	9: F=-1,1 9: E1=0,1 ...
collide1	-	OK	9: F=-1,0 9: E0=0,1 ...
collide2	-	OK	11: F=-1,0 11: E0=...
drop0	-	OK	3: F=-1,2 3: E0=0,1 ...
drop1	-	OK	20: F=-1,2 20: E1=...
drop2	-	OK	3: F=-1,2 3: E1=0,1 ...
fly_over0	-	OK	6: F=-1,1 6: E1=0,1 ...
fly_over1	-	OK	4: F=-1,1 4: E1=0,1 ...
fly_over2	-	OK	9: F=-1,0 9: E0=0,1 ...
follow0	-	OK	7: F=-1,0 7: E0=0,1 ...
follow1	-	OK	7: F=-1,1 7: E0=0,1 ...
follow2	-	OK	7: F=-1,1 7: E0=0,1 ...
give0	-	OK	9: F=-1,2 9: E0=0,1 ...
give1	-	OK	6: F=-1,0 6: E0=0,1 ...
give2	-	OK	4: F=-1,2 4: E1=0,1 ...
hit0	-	OK	30: F=-1,1 30: E1=...

Alpha	Classifier
1.92507...	{[Y0=0,-1, C0,1=0,1, Y0=-1,0]}
1.70612...	{[C0,1=0,1, Y0=-1,0, R0,1=-1,0]}
3.09708...	{[R0,1=0,-1, Y0=-1,0, X0=-1,0]}

Figure A-4: Classifier ensemble for catch



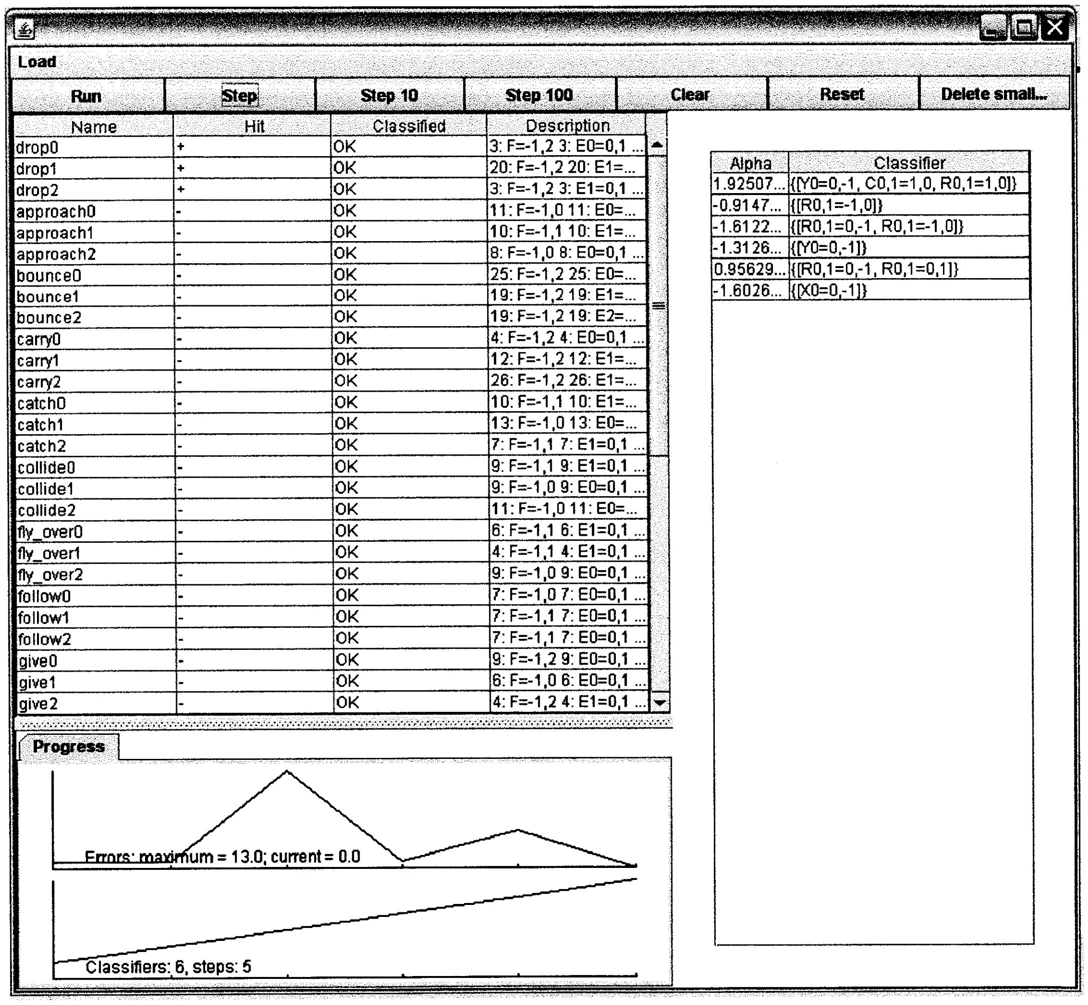
Name	Hit	Classified	Description
collide0	+	OK	9: F=-1,1 9: E1=0,1 ...
collide1	+	OK	9: F=-1,0 9: E0=0,1 ...
collide2	+	OK	11: F=-1,0 11: E0=...
approach0	-	OK	11: F=-1,0 11: E0=...
approach1	-	OK	10: F=-1,1 10: E1=...
approach2	-	OK	8: F=-1,0 8: E0=0,1 ...
bounce0	-	OK	25: F=-1,2 25: E0=...
bounce1	-	OK	19: F=-1,2 19: E1=...
bounce2	-	OK	19: F=-1,2 19: E2=...
carry0	-	OK	4: F=-1,2 4: E0=0,1 ...
carry1	-	OK	12: F=-1,2 12: E1=...
carry2	-	OK	26: F=-1,2 26: E1=...
catch0	-	OK	10: F=-1,1 10: E1=...
catch1	-	OK	13: F=-1,0 13: E0=...
catch2	-	OK	7: F=-1,1 7: E1=0,1 ...
drop0	-	OK	3: F=-1,2 3: E0=0,1 ...
drop1	-	OK	20: F=-1,2 20: E1=...
drop2	-	OK	3: F=-1,2 3: E1=0,1 ...
fly_over0	-	OK	6: F=-1,1 6: E1=0,1 ...
fly_over1	-	OK	4: F=-1,1 4: E1=0,1 ...
fly_over2	-	OK	9: F=-1,0 9: E0=0,1 ...
follow0	-	OK	7: F=-1,0 7: E0=0,1 ...
follow1	-	OK	7: F=-1,1 7: E0=0,1 ...
follow2	-	OK	7: F=-1,1 7: E0=0,1 ...
give0	-	OK	9: F=-1,2 9: E0=0,1 ...
give1	-	OK	6: F=-1,0 6: E0=0,1 ...
give2	-	OK	4: F=-1,2 4: E1=0,1 ...

Errors: maximum = 10.0; current = 0.0

Classifiers: 4, steps: 3

Alpha	Classifier
1.5677...	{[S0=-1,0, Y0=1,0, X1=1,0]}
1.0520...	{[R0,1=-1,0, R0,1=1,0, C0,1=1,0]}
1.8118...	{[R0,1=-1,0, R0,1=0,1, C0,1=1,0]}
-1.632...	{[C0,1=0,1, R0,1=0,-1]}

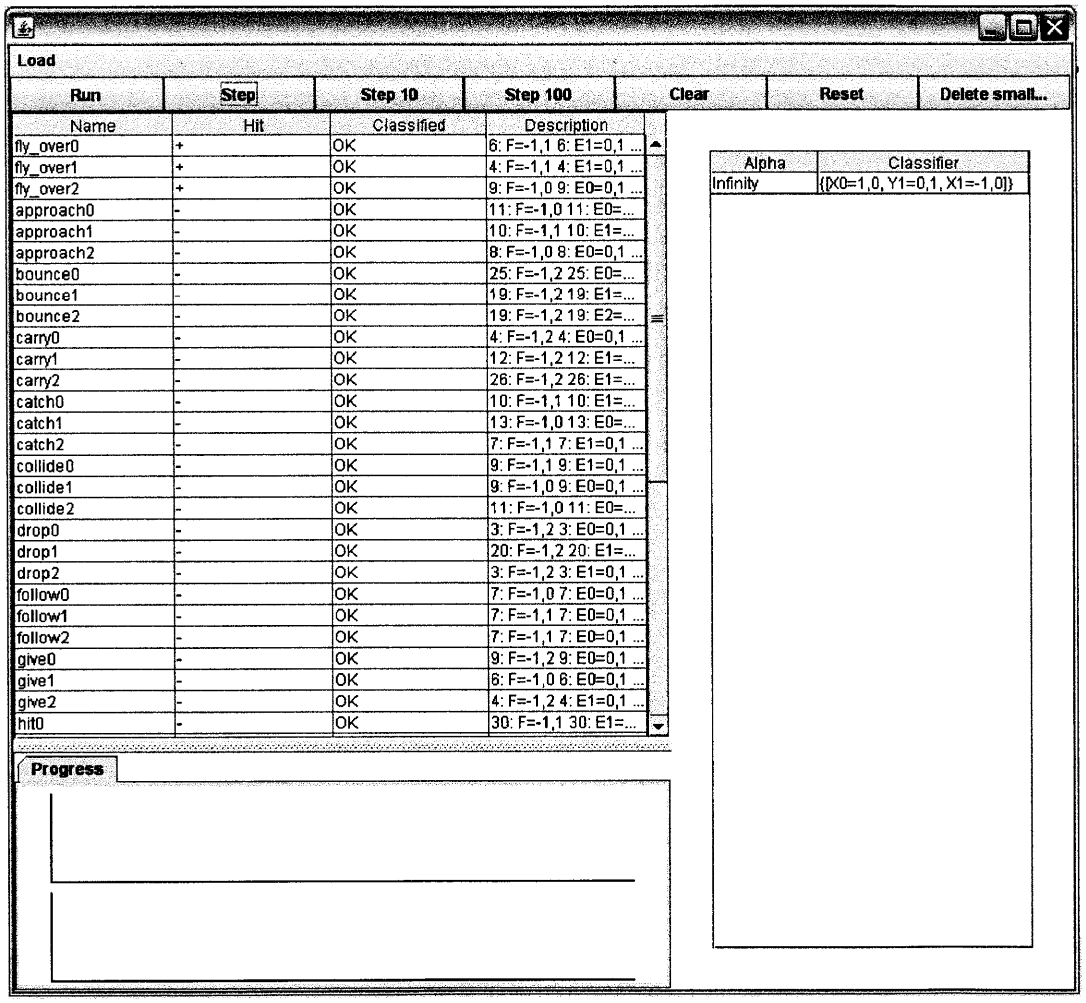
Figure A-5: Classifier ensemble for collide



Name	Hit	Classified	Description
drop0	+	OK	3: F=-1,2 3: E0=0,1 ...
drop1	+	OK	20: F=-1,2 20: E1=...
drop2	+	OK	3: F=-1,2 3: E1=0,1 ...
approach0	-	OK	11: F=-1,0 11: E0=...
approach1	-	OK	10: F=-1,1 10: E1=...
approach2	-	OK	8: F=-1,0 8: E0=0,1 ...
bounce0	-	OK	25: F=-1,2 25: E0=...
bounce1	-	OK	19: F=-1,2 19: E1=...
bounce2	-	OK	19: F=-1,2 19: E2=...
carry0	-	OK	4: F=-1,2 4: E0=0,1 ...
carry1	-	OK	12: F=-1,2 12: E1=...
carry2	-	OK	26: F=-1,2 26: E1=...
catch0	-	OK	10: F=-1,1 10: E1=...
catch1	-	OK	13: F=-1,0 13: E0=...
catch2	-	OK	7: F=-1,1 7: E1=0,1 ...
collide0	-	OK	9: F=-1,1 9: E1=0,1 ...
collide1	-	OK	9: F=-1,0 9: E0=0,1 ...
collide2	-	OK	11: F=-1,0 11: E0=...
fly_over0	-	OK	6: F=-1,1 6: E1=0,1 ...
fly_over1	-	OK	4: F=-1,1 4: E1=0,1 ...
fly_over2	-	OK	9: F=-1,0 9: E0=0,1 ...
follow0	-	OK	7: F=-1,0 7: E0=0,1 ...
follow1	-	OK	7: F=-1,1 7: E0=0,1 ...
follow2	-	OK	7: F=-1,1 7: E0=0,1 ...
give0	-	OK	9: F=-1,2 9: E0=0,1 ...
give1	-	OK	6: F=-1,0 6: E0=0,1 ...
give2	-	OK	4: F=-1,2 4: E1=0,1 ...

Alpha	Classifier
1.92507...	{[Y0=0,-1, C0,1=1,0, R0,1=1,0]}
-0.9147...	{[R0,1=-1,0]}
-1.6122...	{[R0,1=0,-1, R0,1=-1,0]}
-1.3126...	{[Y0=0,-1]}
0.95629...	{[R0,1=0,-1, R0,1=0,1]}
-1.6026...	{[X0=0,-1]}

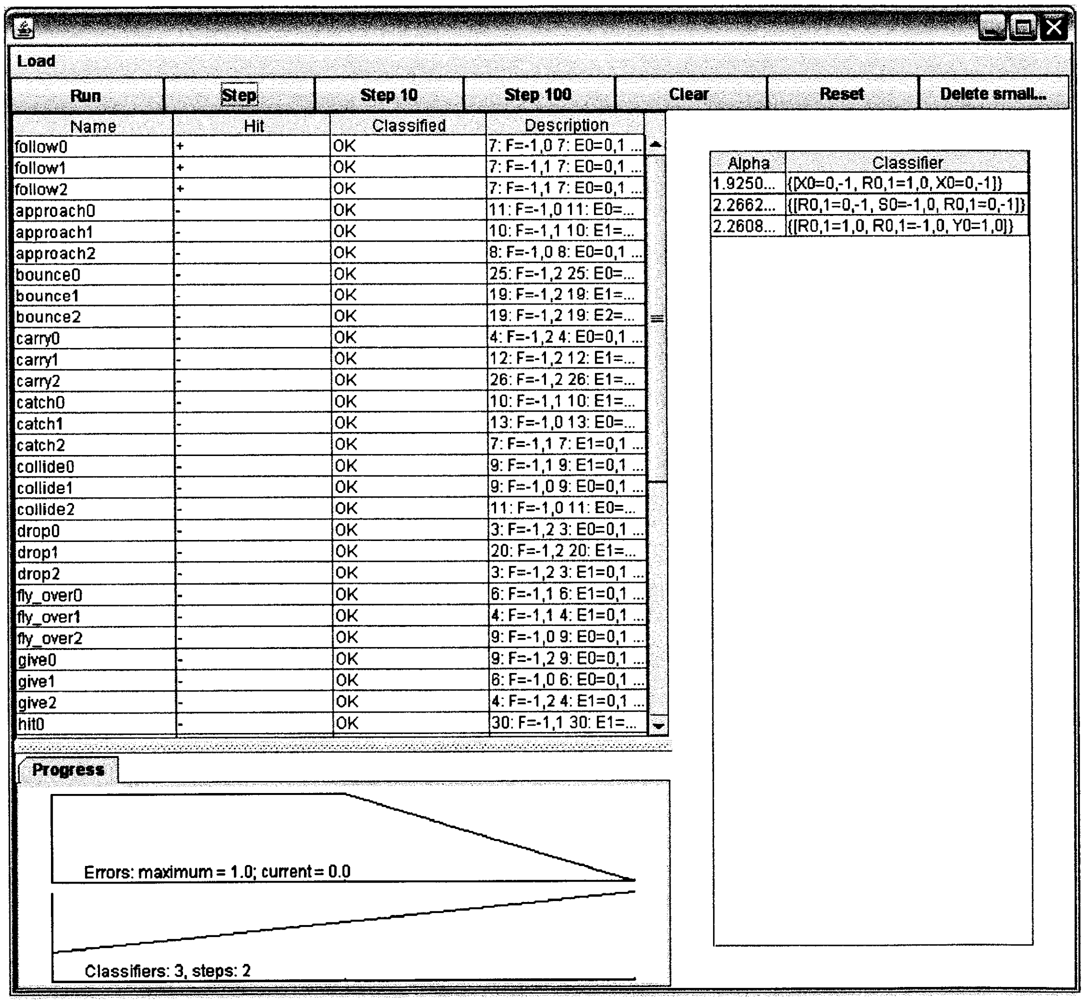
Figure A-6: Classifier ensemble for drop



Name	Hit	Classified	Description
fly_over0	+	OK	6: F=-1,1 6: E1=0,1 ...
fly_over1	+	OK	4: F=-1,1 4: E1=0,1 ...
fly_over2	+	OK	9: F=-1,0 9: E0=0,1 ...
approach0	-	OK	11: F=-1,0 11: E0=...
approach1	-	OK	10: F=-1,1 10: E1=...
approach2	-	OK	8: F=-1,0 8: E0=0,1 ...
bounce0	-	OK	25: F=-1,2 25: E0=...
bounce1	-	OK	19: F=-1,2 19: E1=...
bounce2	-	OK	19: F=-1,2 19: E2=...
carry0	-	OK	4: F=-1,2 4: E0=0,1 ...
carry1	-	OK	12: F=-1,2 12: E1=...
carry2	-	OK	26: F=-1,2 26: E1=...
catch0	-	OK	10: F=-1,1 10: E1=...
catch1	-	OK	13: F=-1,0 13: E0=...
catch2	-	OK	7: F=-1,1 7: E1=0,1 ...
collide0	-	OK	9: F=-1,1 9: E1=0,1 ...
collide1	-	OK	9: F=-1,0 9: E0=0,1 ...
collide2	-	OK	11: F=-1,0 11: E0=...
drop0	-	OK	3: F=-1,2 3: E0=0,1 ...
drop1	-	OK	20: F=-1,2 20: E1=...
drop2	-	OK	3: F=-1,2 3: E1=0,1 ...
follow0	-	OK	7: F=-1,0 7: E0=0,1 ...
follow1	-	OK	7: F=-1,1 7: E0=0,1 ...
follow2	-	OK	7: F=-1,1 7: E0=0,1 ...
give0	-	OK	9: F=-1,2 9: E0=0,1 ...
give1	-	OK	6: F=-1,0 6: E0=0,1 ...
give2	-	OK	4: F=-1,2 4: E1=0,1 ...
hit0	-	OK	30: F=-1,1 30: E1=...

Alpha	Classifier
Infinity	{[X0=1,0, Y1=0,1, X1=-1,0]}

Figure A-7: Classifier ensemble for fly_over



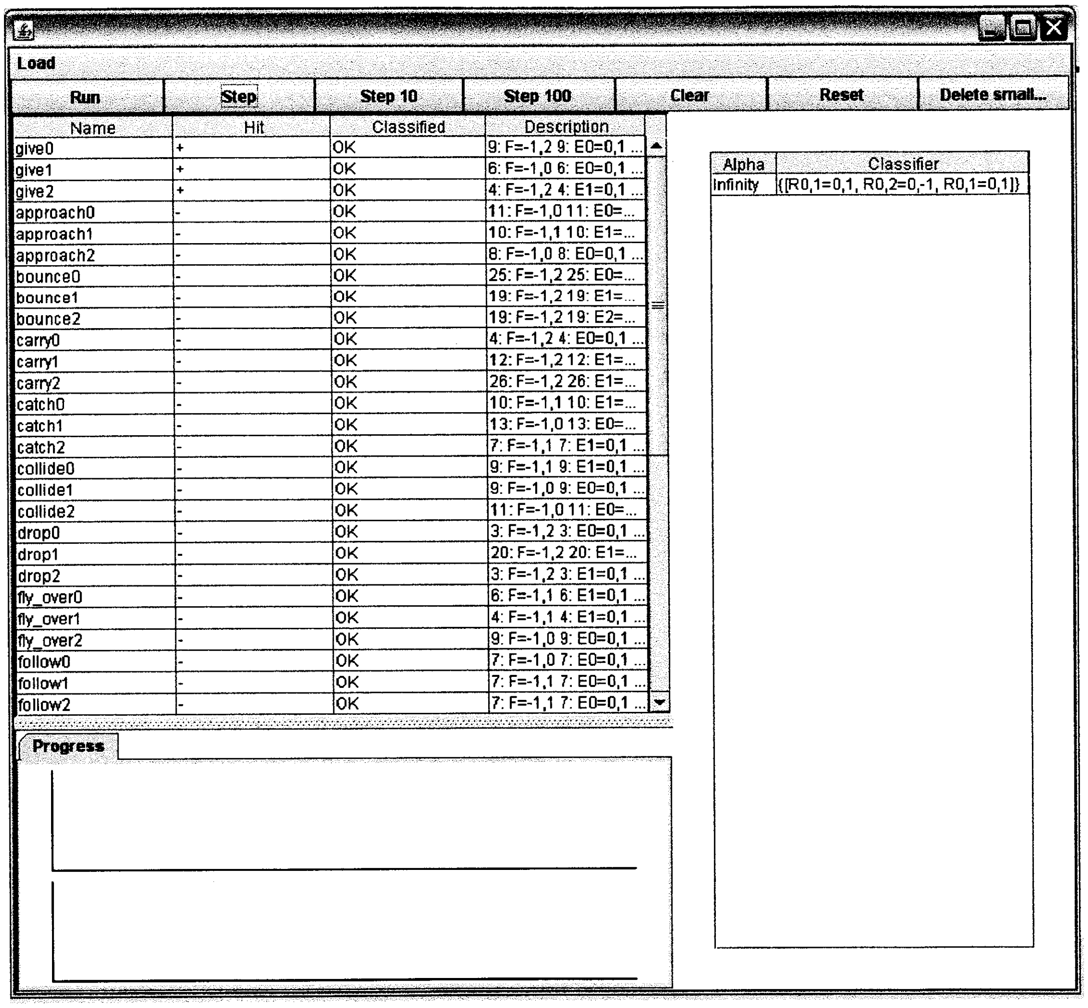
Name	Hit	Classified	Description
follow0	+	OK	7: F=-1,0 7: E0=0,1 ...
follow1	+	OK	7: F=-1,1 7: E0=0,1 ...
follow2	+	OK	7: F=-1,1 7: E0=0,1 ...
approach0	-	OK	11: F=-1,0 11: E0=...
approach1	-	OK	10: F=-1,1 10: E1=...
approach2	-	OK	8: F=-1,0 8: E0=0,1 ...
bounce0	-	OK	25: F=-1,2 25: E0=...
bounce1	-	OK	19: F=-1,2 19: E1=...
bounce2	-	OK	19: F=-1,2 19: E2=...
carry0	-	OK	4: F=-1,2 4: E0=0,1 ...
carry1	-	OK	12: F=-1,2 12: E1=...
carry2	-	OK	26: F=-1,2 26: E1=...
catch0	-	OK	10: F=-1,1 10: E1=...
catch1	-	OK	13: F=-1,0 13: E0=...
catch2	-	OK	7: F=-1,1 7: E1=0,1 ...
collide0	-	OK	9: F=-1,1 9: E1=0,1 ...
collide1	-	OK	9: F=-1,0 9: E0=0,1 ...
collide2	-	OK	11: F=-1,0 11: E0=...
drop0	-	OK	3: F=-1,2 3: E0=0,1 ...
drop1	-	OK	20: F=-1,2 20: E1=...
drop2	-	OK	3: F=-1,2 3: E1=0,1 ...
fly_over0	-	OK	6: F=-1,1 6: E1=0,1 ...
fly_over1	-	OK	4: F=-1,1 4: E1=0,1 ...
fly_over2	-	OK	9: F=-1,0 9: E0=0,1 ...
give0	-	OK	9: F=-1,2 9: E0=0,1 ...
give1	-	OK	6: F=-1,0 6: E0=0,1 ...
give2	-	OK	4: F=-1,2 4: E1=0,1 ...
hit0	-	OK	30: F=-1,1 30: E1=...

Alpha	Classifier
1.9250...	{[X0=0,-1, R0,1=1,0, X0=0,-1]}
2.2662...	{[R0,1=0,-1, S0=-1,0, R0,1=0,-1]}
2.2608...	{[R0,1=1,0, R0,1=-1,0, Y0=1,0]}

Errors: maximum = 1.0; current= 0.0

Classifiers: 3, steps: 2

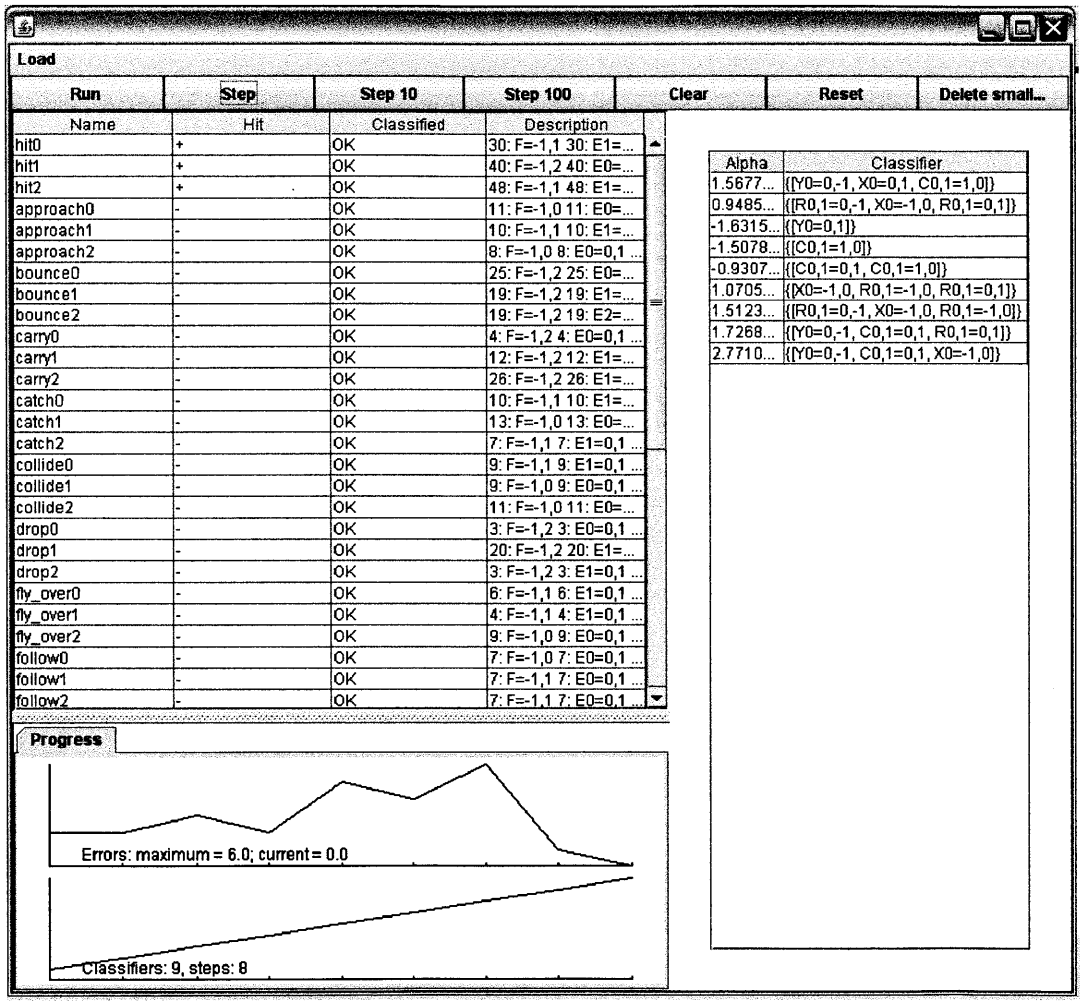
Figure A-8: Classifier ensemble for follow



Name	Hit	Classified	Description
give0	+	OK	9: F=-1,2 9: E0=0,1 ...
give1	+	OK	6: F=-1,0 6: E0=0,1 ...
give2	+	OK	4: F=-1,2 4: E1=0,1 ...
approach0	-	OK	11: F=-1,0 11: E0=...
approach1	-	OK	10: F=-1,1 10: E1=...
approach2	-	OK	8: F=-1,0 8: E0=0,1 ...
bounce0	-	OK	25: F=-1,2 25: E0=...
bounce1	-	OK	19: F=-1,2 19: E1=...
bounce2	-	OK	19: F=-1,2 19: E2=...
carry0	-	OK	4: F=-1,2 4: E0=0,1 ...
carry1	-	OK	12: F=-1,2 12: E1=...
carry2	-	OK	26: F=-1,2 26: E1=...
catch0	-	OK	10: F=-1,1 10: E1=...
catch1	-	OK	13: F=-1,0 13: E0=...
catch2	-	OK	7: F=-1,1 7: E1=0,1 ...
collide0	-	OK	9: F=-1,1 9: E1=0,1 ...
collide1	-	OK	9: F=-1,0 9: E0=0,1 ...
collide2	-	OK	11: F=-1,0 11: E0=...
drop0	-	OK	3: F=-1,2 3: E0=0,1 ...
drop1	-	OK	20: F=-1,2 20: E1=...
drop2	-	OK	3: F=-1,2 3: E1=0,1 ...
fly_over0	-	OK	6: F=-1,1 6: E1=0,1 ...
fly_over1	-	OK	4: F=-1,1 4: E1=0,1 ...
fly_over2	-	OK	9: F=-1,0 9: E0=0,1 ...
follow0	-	OK	7: F=-1,0 7: E0=0,1 ...
follow1	-	OK	7: F=-1,1 7: E0=0,1 ...
follow2	-	OK	7: F=-1,1 7: E0=0,1 ...

Alpha	Classifier
Infinity	{[R0,1=0,1, R0,2=0,-1, R0,1=0,1]}

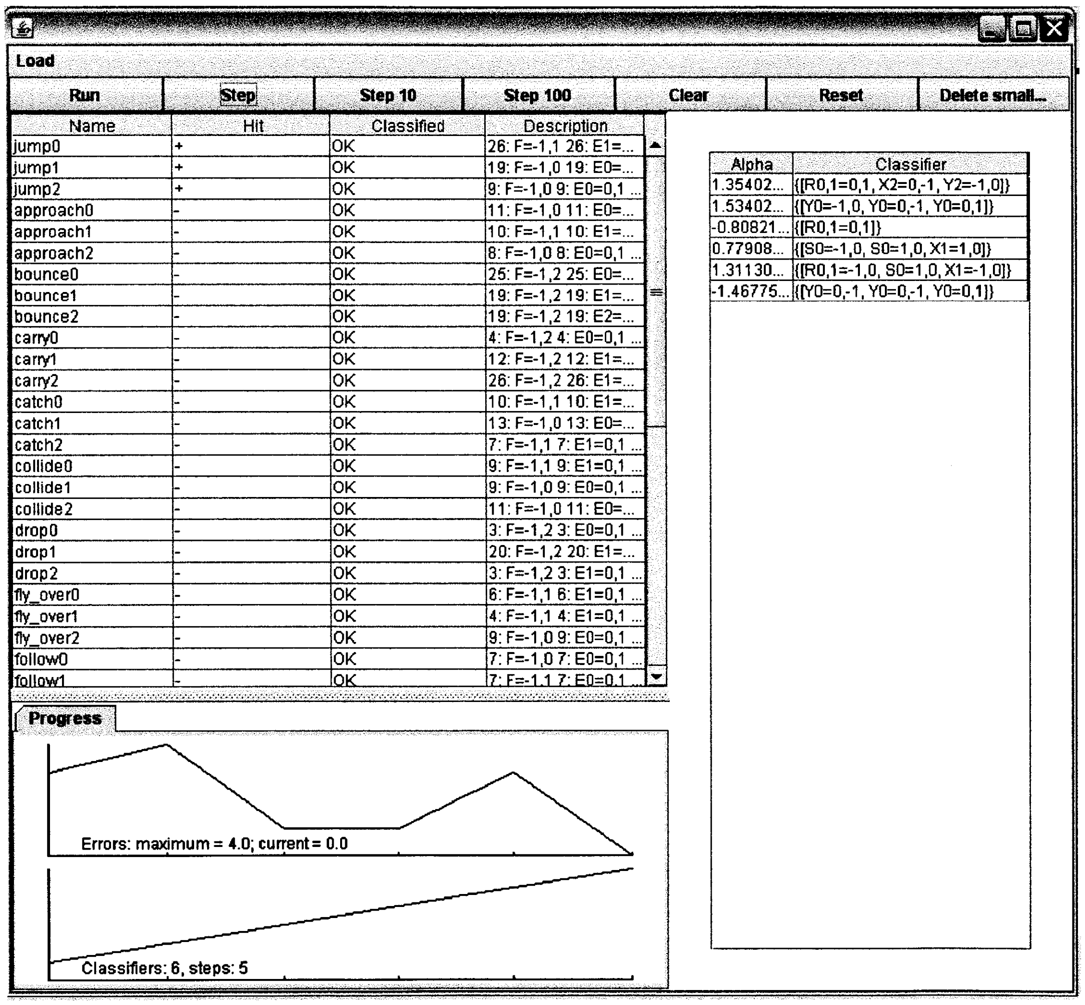
Figure A-9: Classifier ensemble for give



Name	Hit	Classified	Description
hit0	+	OK	30: F=-1,1 30: E1=...
hit1	+	OK	40: F=-1,2 40: E0=...
hit2	+	OK	48: F=-1,1 48: E1=...
approach0	-	OK	11: F=-1,0 11: E0=...
approach1	-	OK	10: F=-1,1 10: E1=...
approach2	-	OK	8: F=-1,0 8: E0=0,1 ...
bounce0	-	OK	25: F=-1,2 25: E0=...
bounce1	-	OK	19: F=-1,2 19: E1=...
bounce2	-	OK	19: F=-1,2 19: E2=...
carry0	-	OK	4: F=-1,2 4: E0=0,1 ...
carry1	-	OK	12: F=-1,2 12: E1=...
carry2	-	OK	26: F=-1,2 26: E1=...
catch0	-	OK	10: F=-1,1 10: E1=...
catch1	-	OK	13: F=-1,0 13: E0=...
catch2	-	OK	7: F=-1,1 7: E1=0,1 ...
collide0	-	OK	9: F=-1,1 9: E1=0,1 ...
collide1	-	OK	9: F=-1,0 9: E0=0,1 ...
collide2	-	OK	11: F=-1,0 11: E0=...
drop0	-	OK	3: F=-1,2 3: E0=0,1 ...
drop1	-	OK	20: F=-1,2 20: E1=...
drop2	-	OK	3: F=-1,2 3: E1=0,1 ...
fly_over0	-	OK	6: F=-1,1 6: E1=0,1 ...
fly_over1	-	OK	4: F=-1,1 4: E1=0,1 ...
fly_over2	-	OK	9: F=-1,0 9: E0=0,1 ...
follow0	-	OK	7: F=-1,0 7: E0=0,1 ...
follow1	-	OK	7: F=-1,1 7: E0=0,1 ...
follow2	-	OK	7: F=-1,1 7: E0=0,1 ...

Alpha	Classifier
1.5677...	{[Y0=0,-1, X0=0,1, C0,1=1,0]}
0.9485...	{[R0,1=0,-1, X0=-1,0, R0,1=0,1]}
-1.6315...	{[Y0=0,1]}
-1.5078...	{[C0,1=1,0]}
-0.9307...	{[C0,1=0,1, C0,1=1,0]}
1.0705...	{[X0=-1,0, R0,1=-1,0, R0,1=0,1]}
1.5123...	{[R0,1=0,-1, X0=-1,0, R0,1=-1,0]}
1.7268...	{[Y0=0,-1, C0,1=0,1, R0,1=0,1]}
2.7710...	{[Y0=0,-1, C0,1=0,1, X0=-1,0]}

Figure A-10: Classifier ensemble for hit



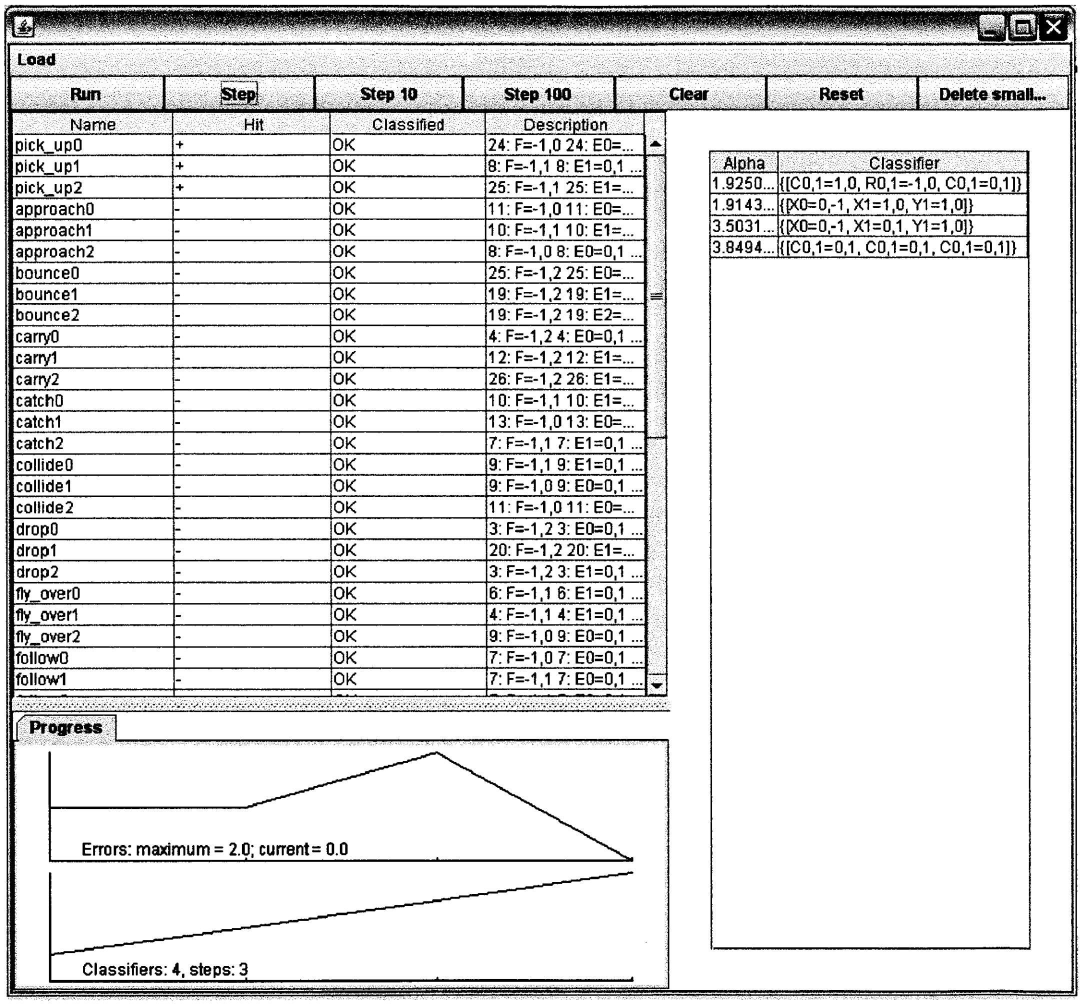
Name	Hit	Classified	Description
jump0	+	OK	26: F=-1,1 26: E1=...
jump1	+	OK	19: F=-1,0 19: E0=...
jump2	+	OK	9: F=-1,0 9: E0=0,1 ...
approach0	-	OK	11: F=-1,0 11: E0=...
approach1	-	OK	10: F=-1,1 10: E1=...
approach2	-	OK	8: F=-1,0 8: E0=0,1 ...
bounce0	-	OK	25: F=-1,2 25: E0=...
bounce1	-	OK	19: F=-1,2 19: E1=...
bounce2	-	OK	19: F=-1,2 19: E2=...
carry0	-	OK	4: F=-1,2 4: E0=0,1 ...
carry1	-	OK	12: F=-1,2 12: E1=...
carry2	-	OK	26: F=-1,2 26: E1=...
catch0	-	OK	10: F=-1,1 10: E1=...
catch1	-	OK	13: F=-1,0 13: E0=...
catch2	-	OK	7: F=-1,1 7: E1=0,1 ...
collide0	-	OK	9: F=-1,1 9: E1=0,1 ...
collide1	-	OK	9: F=-1,0 9: E0=0,1 ...
collide2	-	OK	11: F=-1,0 11: E0=...
drop0	-	OK	3: F=-1,2 3: E0=0,1 ...
drop1	-	OK	20: F=-1,2 20: E1=...
drop2	-	OK	3: F=-1,2 3: E1=0,1 ...
fly_over0	-	OK	6: F=-1,1 6: E1=0,1 ...
fly_over1	-	OK	4: F=-1,1 4: E1=0,1 ...
fly_over2	-	OK	9: F=-1,0 9: E0=0,1 ...
follow0	-	OK	7: F=-1,0 7: E0=0,1 ...
follow1	-	OK	7: F=-1,1 7: E0=0,1 ...

Alpha	Classifier
1.35402...	{[R0,1=0,1, X2=0,-1, Y2=-1,0]}
1.53402...	{[Y0=-1,0, Y0=0,-1, Y0=0,1]}
-0.80821...	{[R0,1=0,1]}
0.77908...	{[S0=-1,0, S0=1,0, X1=1,0]}
1.31130...	{[R0,1=-1,0, S0=1,0, X1=-1,0]}
-1.46775...	{[Y0=0,-1, Y0=0,-1, Y0=0,1]}

Errors: maximum = 4.0; current = 0.0

Classifiers: 6, steps: 5

Figure A-11: Classifier ensemble for jump



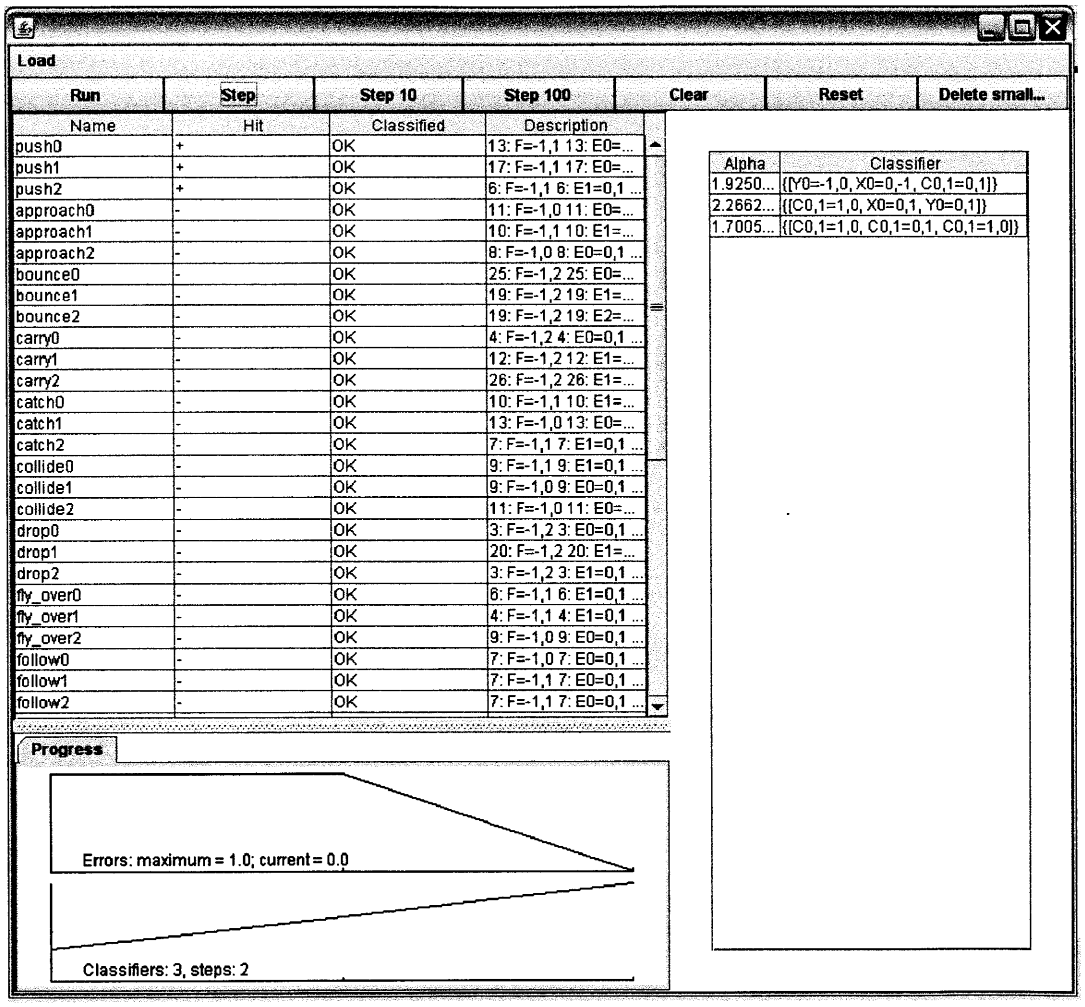
Name	Hit	Classified	Description
pick_up0	+	OK	24: F=-1,0 24: E0=...
pick_up1	+	OK	8: F=-1,1 8: E1=0,1 ...
pick_up2	+	OK	25: F=-1,1 25: E1=...
approach0	-	OK	11: F=-1,0 11: E0=...
approach1	-	OK	10: F=-1,1 10: E1=...
approach2	-	OK	8: F=-1,0 8: E0=0,1 ...
bounce0	-	OK	25: F=-1,2 25: E0=...
bounce1	-	OK	19: F=-1,2 19: E1=...
bounce2	-	OK	19: F=-1,2 19: E2=...
carry0	-	OK	4: F=-1,2 4: E0=0,1 ...
carry1	-	OK	12: F=-1,2 12: E1=...
carry2	-	OK	26: F=-1,2 26: E1=...
catch0	-	OK	10: F=-1,1 10: E1=...
catch1	-	OK	13: F=-1,0 13: E0=...
catch2	-	OK	7: F=-1,1 7: E1=0,1 ...
collide0	-	OK	9: F=-1,1 9: E1=0,1 ...
collide1	-	OK	9: F=-1,0 9: E0=0,1 ...
collide2	-	OK	11: F=-1,0 11: E0=...
drop0	-	OK	3: F=-1,2 3: E0=0,1 ...
drop1	-	OK	20: F=-1,2 20: E1=...
drop2	-	OK	3: F=-1,2 3: E1=0,1 ...
fly_over0	-	OK	6: F=-1,1 6: E1=0,1 ...
fly_over1	-	OK	4: F=-1,1 4: E1=0,1 ...
fly_over2	-	OK	9: F=-1,0 9: E0=0,1 ...
follow0	-	OK	7: F=-1,0 7: E0=0,1 ...
follow1	-	OK	7: F=-1,1 7: E0=0,1 ...

Alpha	Classifier
1.9250...	{[C0,1=1,0, R0,1=-1,0, C0,1=0,1]}
1.9143...	{[X0=0,-1, X1=1,0, Y1=1,0]}
3.5031...	{[X0=0,-1, X1=0,1, Y1=1,0]}
3.8494...	{[C0,1=0,1, C0,1=0,1, C0,1=0,1]}

Errors: maximum = 2.0; current = 0.0

Classifiers: 4, steps: 3

Figure A-12: Classifier ensemble for pick up



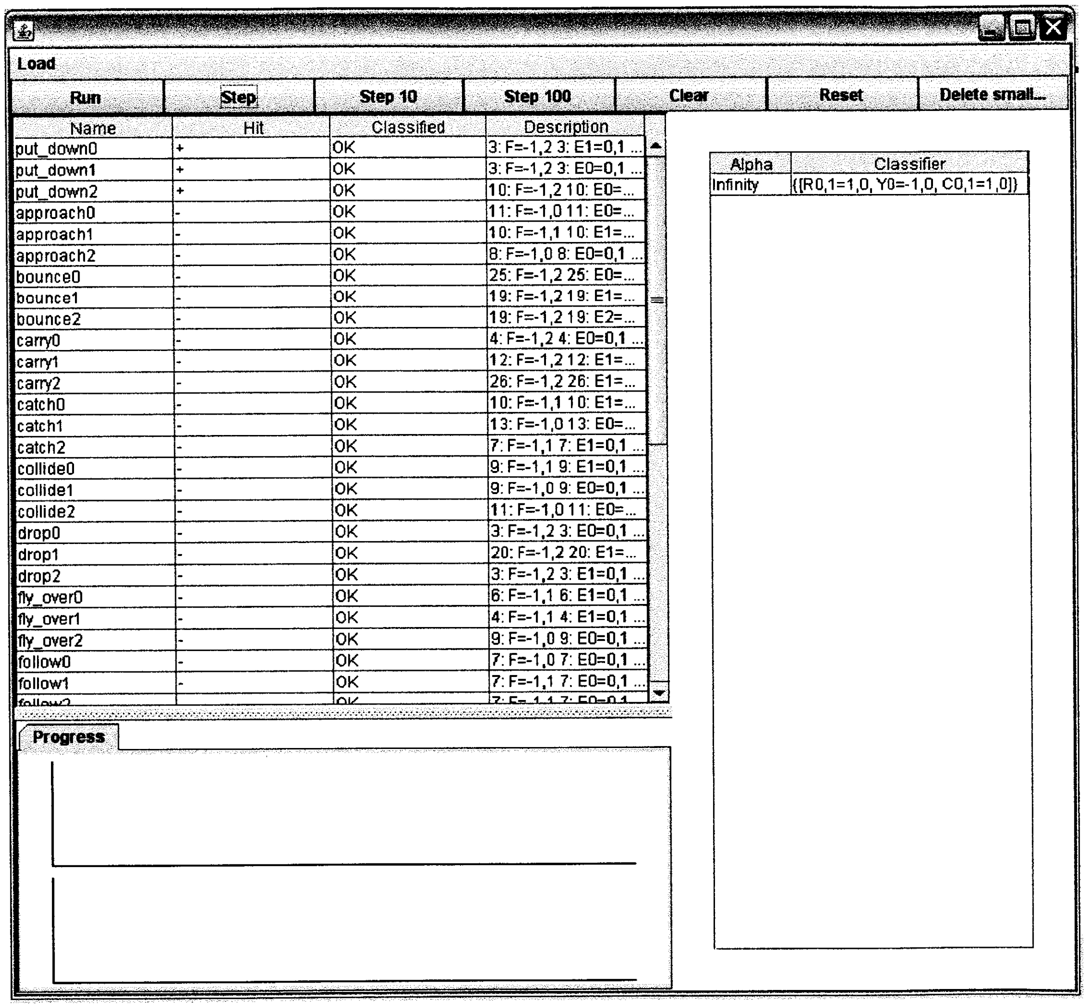
Name	Hit	Classified	Description
push0	+	OK	13: F=-1,1 13: E0=...
push1	+	OK	17: F=-1,1 17: E0=...
push2	+	OK	6: F=-1,1 6: E1=0,1 ...
approach0	-	OK	11: F=-1,0 11: E0=...
approach1	-	OK	10: F=-1,1 10: E1=...
approach2	-	OK	8: F=-1,0 8: E0=0,1 ...
bounce0	-	OK	25: F=-1,2 25: E0=...
bounce1	-	OK	19: F=-1,2 19: E1=...
bounce2	-	OK	19: F=-1,2 19: E2=...
carry0	-	OK	4: F=-1,2 4: E0=0,1 ...
carry1	-	OK	12: F=-1,2 12: E1=...
carry2	-	OK	26: F=-1,2 26: E1=...
catch0	-	OK	10: F=-1,1 10: E1=...
catch1	-	OK	13: F=-1,0 13: E0=...
catch2	-	OK	7: F=-1,1 7: E1=0,1 ...
collide0	-	OK	9: F=-1,1 9: E1=0,1 ...
collide1	-	OK	9: F=-1,0 9: E0=0,1 ...
collide2	-	OK	11: F=-1,0 11: E0=...
drop0	-	OK	3: F=-1,2 3: E0=0,1 ...
drop1	-	OK	20: F=-1,2 20: E1=...
drop2	-	OK	3: F=-1,2 3: E1=0,1 ...
fly_over0	-	OK	6: F=-1,1 6: E1=0,1 ...
fly_over1	-	OK	4: F=-1,1 4: E1=0,1 ...
fly_over2	-	OK	9: F=-1,0 9: E0=0,1 ...
follow0	-	OK	7: F=-1,0 7: E0=0,1 ...
follow1	-	OK	7: F=-1,1 7: E0=0,1 ...
follow2	-	OK	7: F=-1,1 7: E0=0,1 ...

Alpha	Classifier
1.9250...	{[Y0=-1,0, X0=0,-1, C0,1=0,1]}
2.2662...	{[C0,1=1,0, X0=0,1, Y0=0,1]}
1.7005...	{[C0,1=1,0, C0,1=0,1, C0,1=1,0]}

Errors: maximum = 1.0; current = 0.0

Classifiers: 3, steps: 2

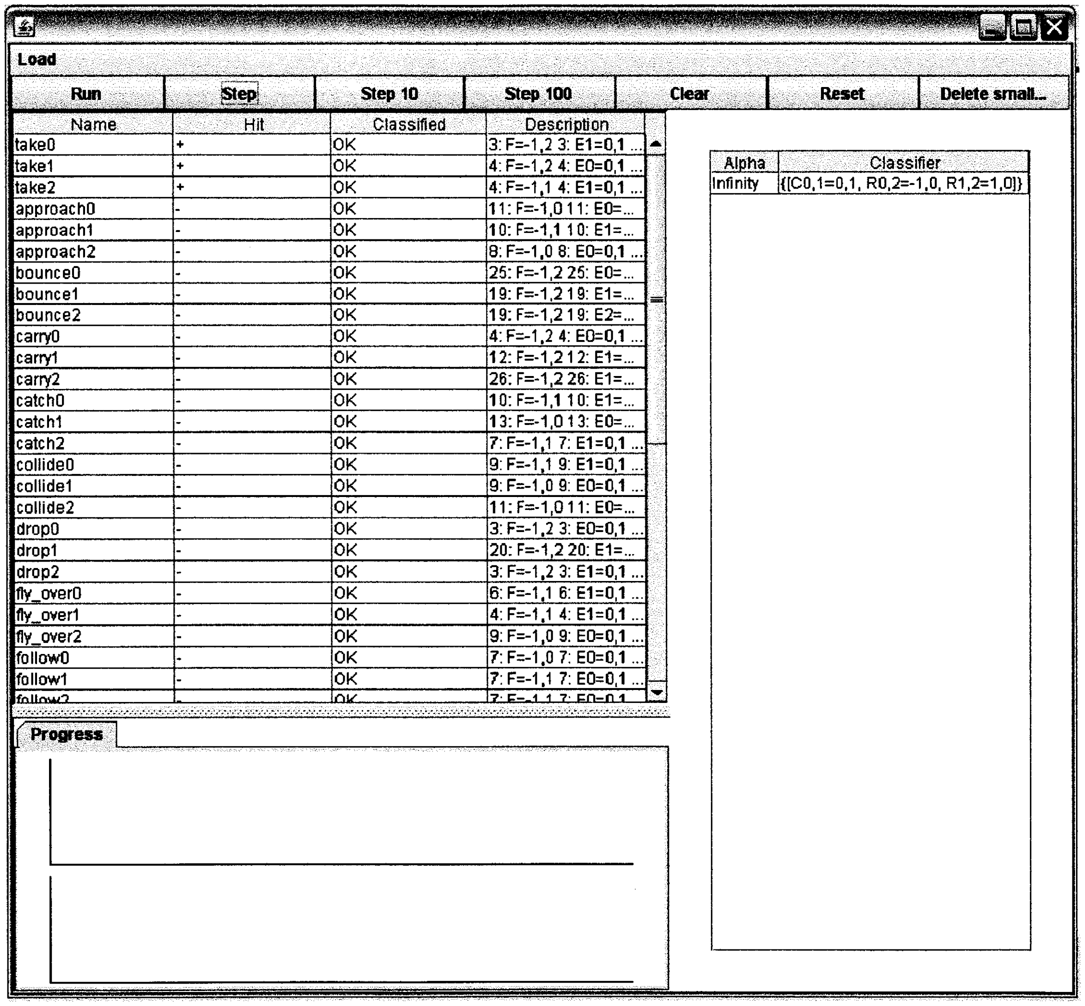
Figure A-13: Classifier ensemble for push



Name	Hit	Classified	Description
put_down0	+	OK	3: F=-1,2 3: E1=0,1 ...
put_down1	+	OK	3: F=-1,2 3: E0=0,1 ...
put_down2	+	OK	10: F=-1,2 10: E0=...
approach0	-	OK	11: F=-1,0 11: E0=...
approach1	-	OK	10: F=-1,1 10: E1=...
approach2	-	OK	8: F=-1,0 8: E0=0,1 ...
bounce0	-	OK	25: F=-1,2 25: E0=...
bounce1	-	OK	19: F=-1,2 19: E1=...
bounce2	-	OK	19: F=-1,2 19: E2=...
carry0	-	OK	4: F=-1,2 4: E0=0,1 ...
carry1	-	OK	12: F=-1,2 12: E1=...
carry2	-	OK	26: F=-1,2 26: E1=...
catch0	-	OK	10: F=-1,1 10: E1=...
catch1	-	OK	13: F=-1,0 13: E0=...
catch2	-	OK	7: F=-1,1 7: E1=0,1 ...
collide0	-	OK	9: F=-1,1 9: E1=0,1 ...
collide1	-	OK	9: F=-1,0 9: E0=0,1 ...
collide2	-	OK	11: F=-1,0 11: E0=...
drop0	-	OK	3: F=-1,2 3: E0=0,1 ...
drop1	-	OK	20: F=-1,2 20: E1=...
drop2	-	OK	3: F=-1,2 3: E1=0,1 ...
fly_over0	-	OK	6: F=-1,1 6: E1=0,1 ...
fly_over1	-	OK	4: F=-1,1 4: E1=0,1 ...
fly_over2	-	OK	9: F=-1,0 9: E0=0,1 ...
follow0	-	OK	7: F=-1,0 7: E0=0,1 ...
follow1	-	OK	7: F=-1,1 7: E0=0,1 ...
follow2		OK	7: F=-1,1 7: E0=0,1

Alpha	Classifier
Infinity	{[R0,1=1,0, Y0=-1,0, C0,1=1,0]}

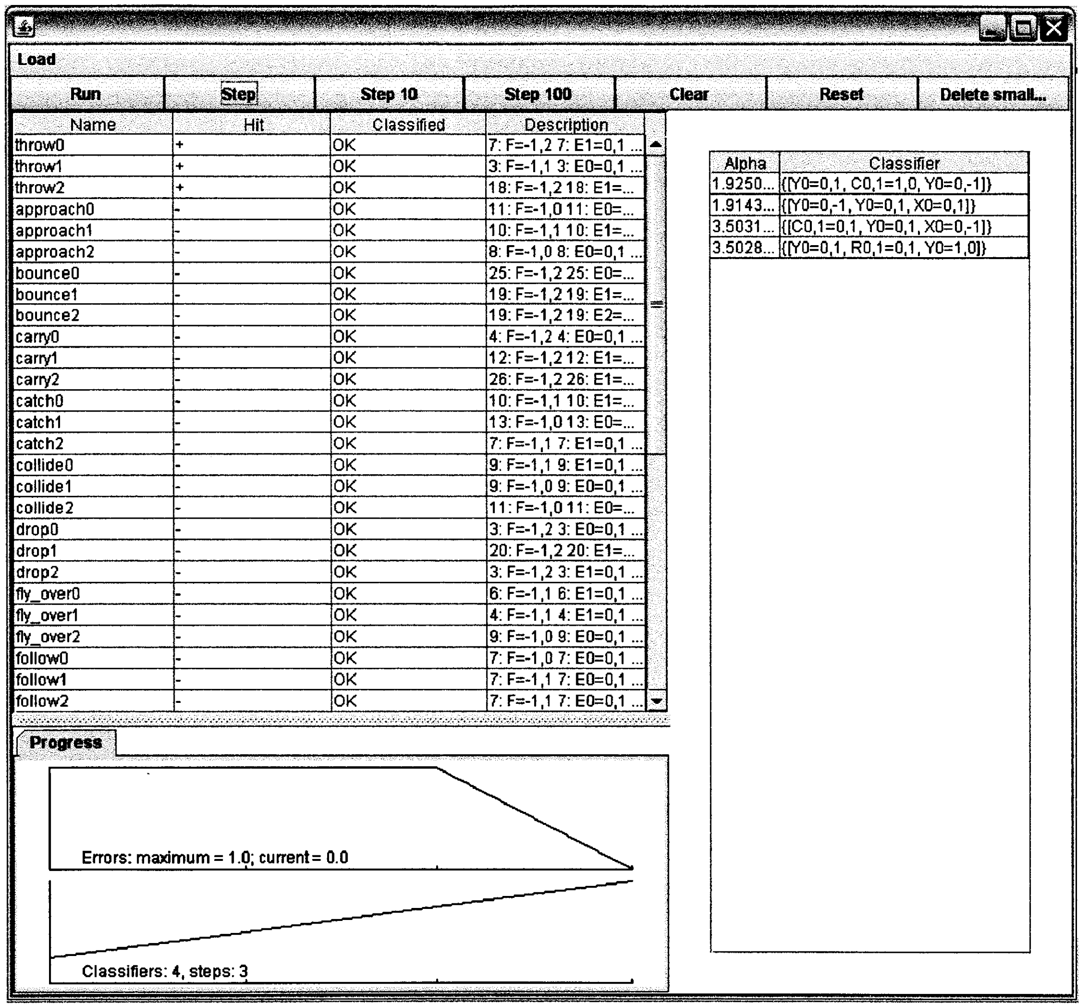
Figure A-14: Classifier ensemble for put down



Name	Hit	Classified	Description
take0	+	OK	3: F=-1,2 3: E1=0,1 ...
take1	+	OK	4: F=-1,2 4: E0=0,1 ...
take2	+	OK	4: F=-1,1 4: E1=0,1 ...
approach0	-	OK	11: F=-1,0 11: E0=...
approach1	-	OK	10: F=-1,1 10: E1=...
approach2	-	OK	8: F=-1,0 8: E0=0,1 ...
bounce0	-	OK	25: F=-1,2 25: E0=...
bounce1	-	OK	19: F=-1,2 19: E1=...
bounce2	-	OK	19: F=-1,2 19: E2=...
carry0	-	OK	4: F=-1,2 4: E0=0,1 ...
carry1	-	OK	12: F=-1,2 12: E1=...
carry2	-	OK	26: F=-1,2 26: E1=...
catch0	-	OK	10: F=-1,1 10: E1=...
catch1	-	OK	13: F=-1,0 13: E0=...
catch2	-	OK	7: F=-1,1 7: E1=0,1 ...
collide0	-	OK	9: F=-1,1 9: E1=0,1 ...
collide1	-	OK	9: F=-1,0 9: E0=0,1 ...
collide2	-	OK	11: F=-1,0 11: E0=...
drop0	-	OK	3: F=-1,2 3: E0=0,1 ...
drop1	-	OK	20: F=-1,2 20: E1=...
drop2	-	OK	3: F=-1,2 3: E1=0,1 ...
fly_over0	-	OK	6: F=-1,1 6: E1=0,1 ...
fly_over1	-	OK	4: F=-1,1 4: E1=0,1 ...
fly_over2	-	OK	9: F=-1,0 9: E0=0,1 ...
follow0	-	OK	7: F=-1,0 7: E0=0,1 ...
follow1	-	OK	7: F=-1,1 7: E0=0,1 ...
follow2	-	OK	7: F=-1,1 7: E0=0,1

Alpha	Classifier
Infinity	{[C0,1=0,1, R0,2=-1,0, R1,2=1,0]}

Figure A-15: Classifier ensemble for take



Name	Hit	Classified	Description
throw0	+	OK	7: F=-1,2 7: E1=0,1 ...
throw1	+	OK	3: F=-1,1 3: E0=0,1 ...
throw2	+	OK	18: F=-1,2 18: E1=...
approach0	-	OK	11: F=-1,0 11: E0=...
approach1	-	OK	10: F=-1,1 10: E1=...
approach2	-	OK	8: F=-1,0 8: E0=0,1 ...
bounce0	-	OK	25: F=-1,2 25: E0=...
bounce1	-	OK	19: F=-1,2 19: E1=...
bounce2	-	OK	19: F=-1,2 19: E2=...
carry0	-	OK	4: F=-1,2 4: E0=0,1 ...
carry1	-	OK	12: F=-1,2 12: E1=...
carry2	-	OK	26: F=-1,2 26: E1=...
catch0	-	OK	10: F=-1,1 10: E1=...
catch1	-	OK	13: F=-1,0 13: E0=...
catch2	-	OK	7: F=-1,1 7: E1=0,1 ...
collide0	-	OK	9: F=-1,1 9: E1=0,1 ...
collide1	-	OK	9: F=-1,0 9: E0=0,1 ...
collide2	-	OK	11: F=-1,0 11: E0=...
drop0	-	OK	3: F=-1,2 3: E0=0,1 ...
drop1	-	OK	20: F=-1,2 20: E1=...
drop2	-	OK	3: F=-1,2 3: E1=0,1 ...
fly_over0	-	OK	6: F=-1,1 6: E1=0,1 ...
fly_over1	-	OK	4: F=-1,1 4: E1=0,1 ...
fly_over2	-	OK	9: F=-1,0 9: E0=0,1 ...
follow0	-	OK	7: F=-1,0 7: E0=0,1 ...
follow1	-	OK	7: F=-1,1 7: E0=0,1 ...
follow2	-	OK	7: F=-1,1 7: E0=0,1 ...

Alpha	Classifier
1.9250...	{[Y0=0,1, C0,1=1,0, Y0=0,-1]}
1.9143...	{[Y0=0,-1, Y0=0,1, X0=0,1]}
3.5031...	{[C0,1=0,1, Y0=0,1, X0=0,-1]}
3.5028...	{[Y0=0,1, R0,1=0,1, Y0=1,0]}

Errors: maximum = 1.0; current = 0.0

Classifiers: 4, steps: 3

Figure A-16: Classifier ensemble for throw

Bibliography

G. C. Borchardt. A computer model for the representation and identification of physical events. Master's thesis, report t-142, University of Illinois, Coordinated Science Laboratory, University of Illinois, Urbana,. IL, 1984.

G. C. Borchardt. Casual reconstruction. A.I. Memo 1403, Massachusetts Institute of Technology, Aritificial Intelligence Laboratory, Cambridge, Massachusetts, February 1993.

Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting, 1995.

Satyajit Rao. *Visual Routines and Attention.* PhD dissertation, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science, February 1998.

Patrick Henry Winston. Learning by analyzing differences. In *Artificial intelligence (3. ed.)*, pages 349–363. Addison-Wesley, 1992. ISBN 978-0-201-53377-4.