Internal Use Only (非公開)

TR-I-0302

Manual for Speech Input Programs: SpeechIn, SpeechPress

Harald Singer, Masahide Sugiyama

1993.3

ABSTRACT

This report describes SpeechIn and SpeechPress, two X Window System based programs for real-time input and automatic detection of speech. The programs were designed as frontend for demonstration systems at ATR. The main differences to its predecessors segment and segmentNew are

1. automatic start and endpoint detection
2. improved user interface
3. support for several different configurations, i.e. DEC (ULTRIX4.1) with SCSI bus, DEC (ULTRIX4.2) with Turbo Channel, DEC (ULTRIX4.2a) with SCSI bus, DEC (ALPHA) with SCSI bus, SUN (SunOS 4.1.1) with VME bus, SUN (SunOS 4.1.1) with S bus, HP (HP-UX 8.05) with Turbo Channel and HP (HP-UX 8.05) with Turbo Channel (new DASBOX).

©ATR 自動翻訳電話研究所
©ATR Interpreting Telephony Research Labs.

Contents

Chapter 1

Introduction

This report describes `SpeechIn` and `SpeechPress`, two X Window System based programs for real-time input and automatic detection of speech. These programs were designed as frontend for demonstration systems at ATR. They also allows "manual" editing of the boundary placements. Such corrections can be made using a pointer device (mouse). Reading-in data from a file instead of requesting data from the AD converter is also supported.

The program was originally developed by several people (see Chapter 5) on a DECstation 3100 connected to a DASBOX 12. It was then ported to a SUN Sparc station and then to a HP 9000/750 with a DASBOX 16. We then did a major rewriting. The same program runs currently on four different architectures, i.e. DEC, DEC ALPHA, SUN and HP.

Chapter 2 explains usage of the program and gives some examples. In chapter 3 we describe the underlying algorithm and some implementation details. Finally, chapter 4 explains installation and update procedures.

At ATR, the files described in this report and the documentation can be found on the machine `atr-fs` under `/NFS/atr-fs/pub1/common/SpeechIn`. Binary executables for each architecture are (and will be) accessible via `/NFS/atr-fs/pub1/common/bin/SpeechIn` and `/NFS/atr-fs/pub1/common/bin/SpeechPress` independent of machine architecture.

Please send comments and bug reports to `singer@itl.atr.co.jp`.

Keywords: speech input, endpoint detection, continuous AD.

Chapter 2

Program Usage for SpeechIn

2.1 Default Usage

The following usage message shows the defaults which you can change on the command line. [] indicates optional arguments. All arguments are optional. <> indicates a required value. For example -hp 0 changes the size of the power window from 50 to 0. {} denotes a toggle flag. For example, simply by specifying -a the value of the *fully automatic* flag changes to YES.

```
usage: SpeechIn <SpeechIn1.14 1993/03/08>
[-a {fully automatic}]       default: NO
[-A <additional margin>]     default: 0
[-b {bunsho segmentation}]   default: NO
[-c <chmod flag file>]       default: <>
[-D <debug level>]           default: 0
[-e {enable manual edit}]    default: YES
[-E <endslots>]              default: 10
[-f {forced exit}]           default: NO
[-F {focus flag}]            default: YES
[-i <input file>]            default: <>
[-I {iconify}]               default: NO
[-ht <text height pixel>]    default: 40
[-hw <wave height pixel>]    default: 230
[-hp <power height pixel>]   default: 50
[-hz <zerocross height pix>]default: 50
[-l {show level}]            default: YES
[-L {lock}]                  default: NO
[-m <add. margin in frames>]default: 0
[-M {minimal window}]        default: NO
[-n {no number for outfile}]default: NO
[-N {no subwindow}]          default: YES
[-o {writing to stdout}]     default: NO
[-O <labelfile>]             default: <>
[-s <wait secs for display>]default: 0
[-S <suffix number>]         default: 0
[-u <unseg. outputfile>]     default: <>
[-v <verbose flag>]          default: YES
[-w <outputfile>]            default: xxx
[-x <x position in pixel>]   default: 100
[-y <y position in pixel>]   default: 100
```

-a	`flag`	exit program *automatically* after first successful segmentation
-A	num	additional margin in data points at beginning of segments
-b	`flag`	whole utterance segmentation ("sentence"): don't cut into phrases (*bunsetsu*) but use whole sentence. This is also useful for speaker adaptation using words.
-c	`file`	file name which is used for sync with translation process; `file`'s mode is changed to 222; `file.nph` for number of phrases after successful segmentation
-D	num	display debug information depending on num: the bigger num is, the more debug information is displayed, i.e if num is 0 no debug information is displayed.
-e	`flag`	enable correction of segmentation boundaries with the mouse
-E	num	minimum number of slots (100ms unit) for judgement of end of utterance
-f	`flag`	force exit
-F	`flag`	grab focus
-i	`file`	read from `file` instead of using data from AD/DA converter
-I	`flag`	iconify window after segmentation
-ht	num	height of text in pixels
-hw	num	height of wave window in pixels
-hp	num	height of power window in pixels
-hz	num	height of zero-cross window in pixels
-l	`flag`	display level meter
-L	`flag`	start in locked mode
-m	num	num additional frames at front and end of each phrase (frameshift is 10 mS); not used for DA!
-M	`flag`	minimize window size
-n	`flag`	don't append a number to the output file names for phrase level segmentation
-N	`flag`	subwindow display (not implemented)
-o	`flag`	sending data to stdout and not to a file; data is preceded by a header (see 2.3)
-O	`file`	output (dummy) labels in ATR format to `file`
-s	num	sleep for num seconds after displaying wave and segmentation boundaries
-S	num	start output with `filenum`, e.g `foo28, foo29` ... if num was set to 28. This is useful for speaker adaptation using words.
-u	`file`	output to `file` of unsegmented data, i.e. only rough endpoint segmentation has been performed (see Chapter 4.1)
-v	`flag`	show program name and version on wave window
-w	`file`	wave data segmented at the phrase level (*bunsetsu*) is written to files `file0`, `file1` etc. (unless -n option was chosen)
-x	num	upper left corner x-coordinate of window in pixels
-y	num	upper left corner y-coordinate of window in pixels

On startup the program measures the surrounding noise level. This noise level is displayed in blue in the left bar of the level meter in the upper left corner of the screen (see Fig. 2.1). The current input level is displayed in green in the right bar. If the input level exceeds the noise level, the input wave is displayed in real time from left to right. After finding the endpoint of

the whole utterance (defined as 1.5 seconds without speech) the data is displayed again with phrase segmentations marked (see Fig. 2.2).

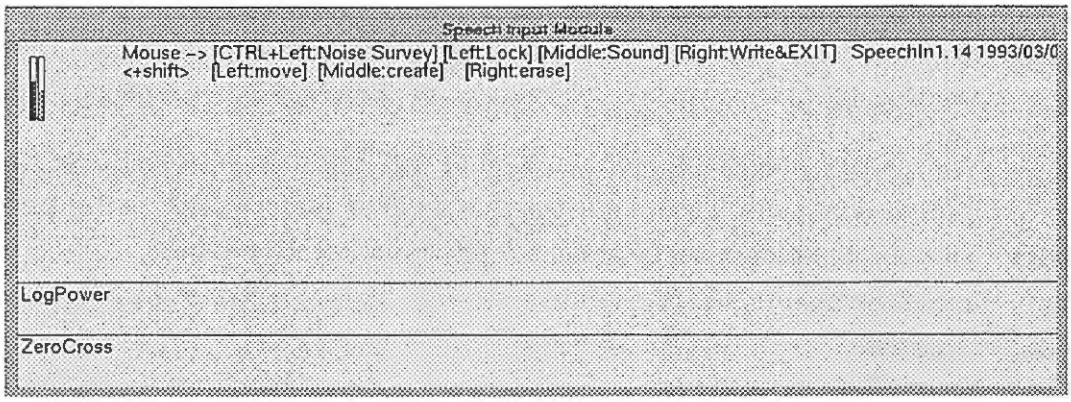

Figure 2.1: Display before utterance

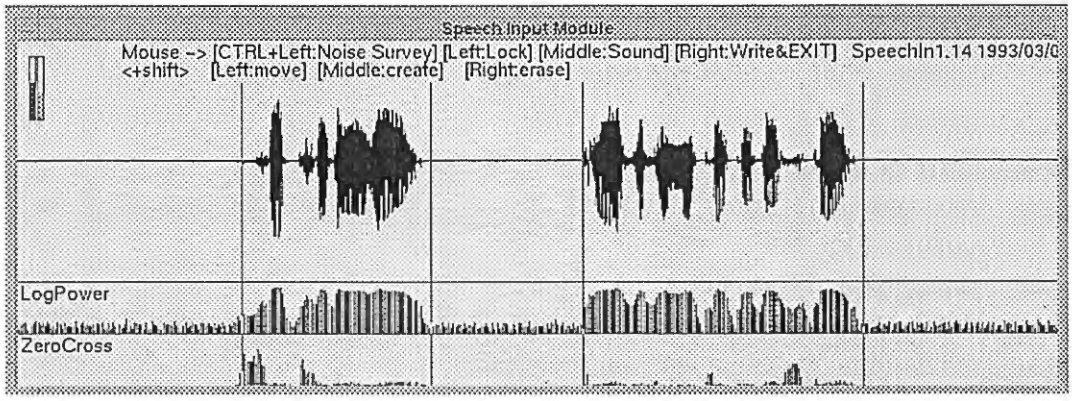

Figure 2.2: Display after end of utterance

The program will continue monitoring the input and checking if new speech is input. If the noise level threshold is exceeded, the previously segmented data is discarded and the new data is read in and segmented. Deactivation and reactivation of this threshold mechanism (locking) is toggled with the LEFT MB (mouse button). The color of the left level bar is changed to red and the sign *LOCK* appears on top of the level bar to display the locked status. Note: changing the locking mechanism is automatically disabled after the noise level threshold has been exceeded until the end of the utterance!

By clicking the MIDDLE MB, the phrase whose boundaries are closest to the cursor position is played back through the DA converter.

The noise level threshold can be updated by clicking the LEFT MB while holding down the CONTROL key.

If the *manual edit* flag is enabled (-e option), the segmentation boundaries can be shifted (SHIFT + LEFT MB), erased (SHIFT + RIGHT MB) or new segmentation boundaries can be added (SHIFT + MIDDLE MB).

Pushing the RIGHT MB causes the program to write the segmented data to disk and exit. The program returns the number of segmented phrases or 0 if no phrase boundaries were found.

If the *fully automatic* flag is enabled (`-a` option), the program usually exits after its first successful endpoint detection. Pushing any MB allows you to abort this exit operation. The program stays in *fully automatic* mode, but requests new speech input from the user.

If you want to exit the program without writing any files to disk, but you already input some speech data, push SHIFT + RIGHT MB until no more boundary lines are displayed and then push RIGHT MB to exit. This works only if the *manual edit* flag is enabled.

2.2 Examples

In the following we will show some of the most often used command options.

- `$ SpeechIn -w foo`

  Phrase segmented output is written to files `foo0`, `foo1` etc..

- `$ SpeechIn -w foo -a -s 2`

  Phrase segmented output is written to files `foo0`, `foo1` etc., exit without query after getting first utterance, wait for 2 seconds before exiting.

- `$ SpeechIn -w foo -L -a -s 2`

  Phrase segmented output is written to files `foo0`, `foo1` etc., start in locked mode, exit without query after getting first utterance, wait for 2 seconds before exiting.

- `$ SpeechIn -w foo -u unseg`

  Phrase segmented output is written to files `foo0`, `foo1` etc., unsegmented output is written to file unseg.

  `$ SpeechIn -w foo -i unseg`

  Phrase segmented output is written to files `foo0`, `foo1` etc., input is read from file unseg.

The following shell script shows how to combine `SpeechIn` with a recognition program via `xmenu` and `xinput` [3].

demonstration.csh

```
#!/bin/csh -f
    set OPTION =
    set JUMP = SPEECHIN
CONTROL:
    xmenu \
    -w demox -n 1 -l -c -x 0 -y 0 -m 60 -v 10 \
   -f -adobe-times-bold-r-normal--34-240-100-100-p-177-iso8859-1 \
   -ct 'red' -t ' ATR Interpreting Telephony' \
   'CONTINUE'\
   'AUTOMATIC INPUT'\
   'SEMI-AUTOMATIC INPUT'\
   'OTHER OPTIONS'\
   'QUIT'

    set ANSWER = $status
    switch( $ANSWER )
    case 1:
      breaksw
    case 2:
      set JUMP = SPEECHIN
      set OPTION = ( -a -s 2 )
      breaksw
    case 3:
      set JUMP = CONTROL
      set OPTION =
      breaksw
    case 4:
      set OPTION = `xinput -y 100 -n 20 -f Helvetica24 -l -t 'Input OPTIONS'`
      breaksw
    case 5:
      exit 0
      breaksw
  endsw
SPEECHIN:
  set NUM = `SpeechIn $OPTION -w xxx`
  if( $NUM == 0 ) goto CONTROL
RECOGNITION:
  @ I = 0
  while($I < $NUM)
        echo "try to recognize xxx$I"
        sleep 2
        echo "show recognition results"
        sleep 2
        @ I = $I + 1
  end
  goto $JUMP
END:
  exit 0
### EOF
```

2.3 Format if Writing to *stdout*

In demo speech recognition systems at ATR, we recently used the filter paradigm: speech is sampled, passed through several cascaded "filters" and finally a string is put out as result of the filtering process. In other words, speech recognition is regarded as a filtering problem with speech as input and recognized strings as output.

Filter connections are realized as UNIX pipes. We therefore added the -o option to the programs to send raw data with an header to the next filter instead of writing to a file. This next filter is usually a program that converts the wave samples to a sequence of 34-dimensional cepstrum and Δcepstrum vectors (e.g. `WavePara34`). The header is defined as follows:

header.h

```
typedef struct {
  int size;     /* size in bytes (without header) */
  int utt;      /* number of bunsetsu (start with 0) */
  int totalutt; /* total number of bunsetsu in bunsho */
  int time1;    /* time value */
  int time2;    /* time value (for future use) */
} ATRHEADER;
```

That is, the binary data in big-endian format `short` is preceded by a 20 byte header (5 * 4 `int`). Consider for example the utterance "kochirawa # kaigizimukyokudesu" (# denoting a phrase boundary). The header of the first phrase contains `utt=0` and `totalutt=2` (0 of 2), the header of the second phrase contains `utt=1` and `totalutt=2` (1 of 2).

`size` contains the number of bytes of the "raw" data.

`time1` was created, using the time subroutine, which returns the time since 00:00:00 GMT, Jan. 1, 1970, measured in seconds. It is used for measuring real-time performance.

SpeechIn can thus be used as follows:

```
$ SpeechIn -o | WavePara34 | Recognize &
```

2.4 Bugs and Problems

- If additional noise enters the microphone during noise level survey (at startup or with CTRL LEFT MB), the segmentation algorithm will not work properly. This is due to the ring buffer implementation. Wait at least 1.5 seconds after noise survey before starting with speech input
- It would be desirable to have similar low-level routines; currently, the DEC and the SUN/HP version differ considerably in the implementation of AD and DA routines.
- For future versions a standard X interface should be used, XG.c should be rewritten, and segment.c should be split in functional parts.
- The DEC version under ULTRIX 4.1 using SCSI bus doesn't work properly on some machines and can cause machine hang-up.
- For ULTRIX 4.2a, several paramters like BUS, ID are used through enviroment variable DASBOX.

  ```
  setenv DASBOX #1 #2 #3
  #1: SCSI BUS No.
  #2: SCSI No.
  #3: LUN ( Local Unit No. ) normally LUN =0.
  ```

  For example with BUS No=0, SCSI No=4 the following command must be used:

  ```
  setenv DASBOX 040
  ```

Chapter 3

Program Usage for SpeechPress

3.1 Default Usage

The following usage message shows the defaults which you can change on the command line. See chapter 2.1 for a detailed explanation of all options. Some of the options don't make sense for SpeechPress and are only kept for compatibility with SpeechIn.

```
usage: SpeechPress <SpeechPress1.11 1993/03/01>
[-a {fully automatic}]       default: NO
[-A <additional margin>]     default: 0
[-b {bunsho segmentation}]   default: NO
[-c <chmod flag file>]       default: <>
[-d {append date to unseg}]  default: NO
[-D <debug level>]           default: 0
[-e {enable manual edit}]    default: YES
[-E <endslots>]              default: 10
[-f {forced exit}]           default: NO
[-F {focus flag}]            default: YES
[-i <input file>]            default: <>
[-I {iconify}]               default: NO
[-H {suppress messages}]     default: NO
[-ht <text height pixel>]    default: 40
[-hw <wave height pixel>]    default: 230
[-hp <power height pixel>]   default: 50
[-hz <zerocross height pix>]default: 50
[-l {show level}]            default: YES
[-L {lock}]                  default: NO
[-m <add. margin in frames>]default: 0
[-M {minimal window}]        default: NO
[-n {no number for outfile}]default: NO
[-N {no subwindow}]          default: YES
[-o {writing to stdout}]     default: NO
[-O <labelfile>]             default: <>
[-s <wait secs for display>]default: 0
[-S <suffix number>]         default: 0
[-u <unseg. outputfile>]     default: <>
[-v <verbose flag>]          default: YES
[-w <outputfile>]            default: xxx
[-x <x position in pixel>]   default: 100
```

```
[-y <y position in pixel>]  default: 100
```

Upon starting the program the initial help screen will be displayed (see Fig. 3.1) unless the *verbose flag* is disabled (-v option). Make sure, that the cursor is in the wave window before speaking! Hold down the LEFT MB (mouse button), start speaking with adequate pauses between phrases, stop speaking and release the LEFT MB. The program calculates segmentation boundaries according to smoothed log-power and zero-crossings and displays them (see Fig. 3.2). Pushing the RIGHT MB causes the program to write the segmented data to disk or standard out (*stdout flag* is enabled with -o option).

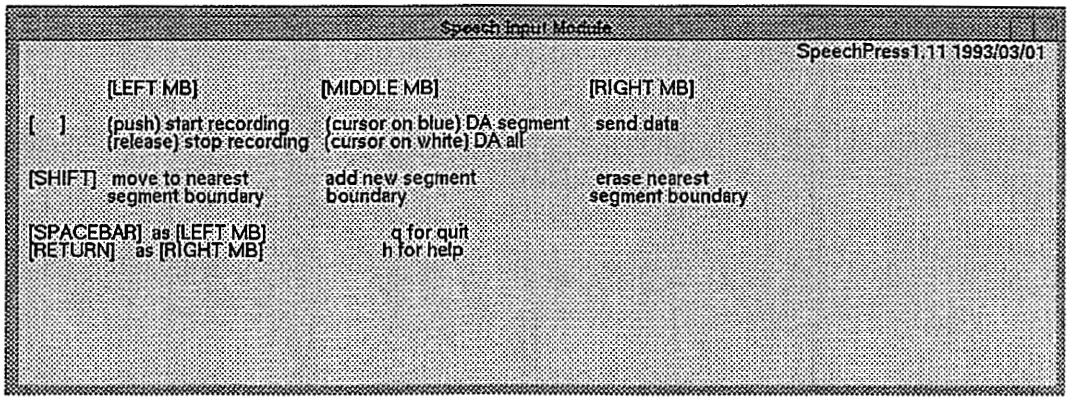

Figure 3.1: Start-up and help screen for SpeechPress

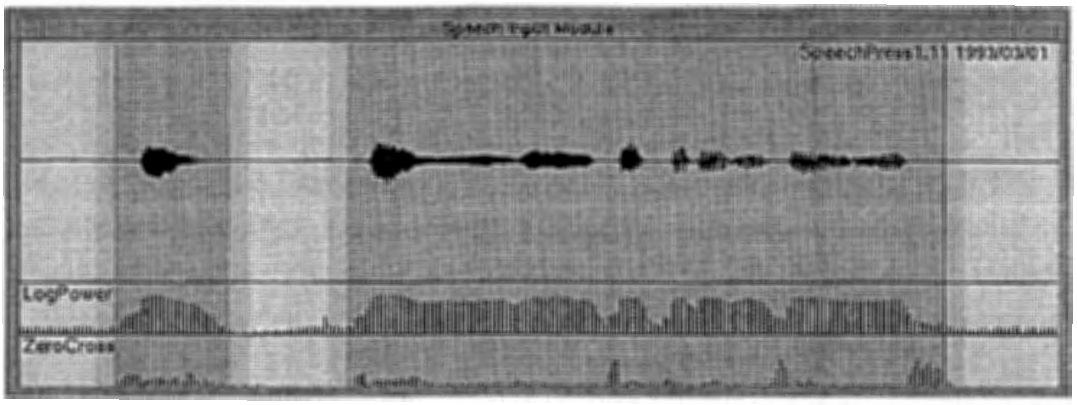

Figure 3.2: Two-phrase input with segmentation marks in SpeechPress

While speaking the wave signal is displayed left to right on the wave window. After 12.5 seconds of input the wave signal reaches the right edge of the screen, the screen is cleared and the wave signal "wraps around" to the left edge of the screen. The maximum length of overall speech including pauses is set to 50 seconds. No noise level threshold is used.

Two levels of shading are used. The darker shading denotes the speech region as found by the extraction algorithm, the lighter shading shows an additional safety margin of about 50 ms

around the speech region [1].

By clicking the MIDDLE MB, the phrase "under the cursor" is played back thro converter. If there is no segmented speech under the cursor, the whole utteran pauses is played back.

To correct false segmentation boundaries, e.g. at a weak /shi/ at the beginning of the segmentation boundaries can be edited. If the *manual edit* flag is enabled (e segmentation boundaries can be shifted (SHIFT + LEFT MB), erased (SHIFT + R or new segmentation boundaries can be added (SHIFT + MIDDLE MB).

The initial help screen can be displayed at any time by holding down h on the keyb want to exit the program while *stdout flag* is enabled, enter q on the keyboard.

[1] Usually, during recognition, an HMM silence model is concatenated in front and at the en strings.

Chapter 4

Algorithm and Implementation Details

SpeechIn performs a rough endpoint detection using absolute values of the waveform data. SpeechInand SpeechPress perform a fine phrase segmentation using smoothed log power and zero-crossings.

4.1 Rough Endpoint Detection (SpeechIn only)

The endpoint detection algorithm is very primitive but fast. Data is requested in *slots* from the AD converter, where a slot contains 1200 points, i.e. 0.1 seconds at 12 kHz. On startup the mean absolute value of 3 slots is measured and a noise level threshold calculated.

Then, every 0.1 seconds a new slot is requested and written to a ring buffer. If the threshold is exceeded, the ring buffer contents are copied to the main buffer. If the mean absolute value of 15 consecutive slots (1.5 seconds) is below the noise level threshold, the algorithm decides that an end of utterance has been found and stops requesting slots. The main wave buffer is passed to the next stage, the fine segmentation algorithm.

As an example, lets suppose that the threshold is exceeded at slot 4 in the ring buffer. The algorithm then fills up the main buffer starting with slot 13. After detecting an endpoint (15 slots with input level below noise level) the contents of the ring buffer are copied into the first 12 slots of the main buffer as depicted in Fig. 4.1.

The last 12 slots (1.2 seconds) of the previously detected utterance, which are supposed to contain only noise, are then copied to the ring buffer for reinitialization. We can't simply set the ring buffer to zero, as the fine segmentation algorithm uses the minimum of the whole detected utterance and thus would become confused.

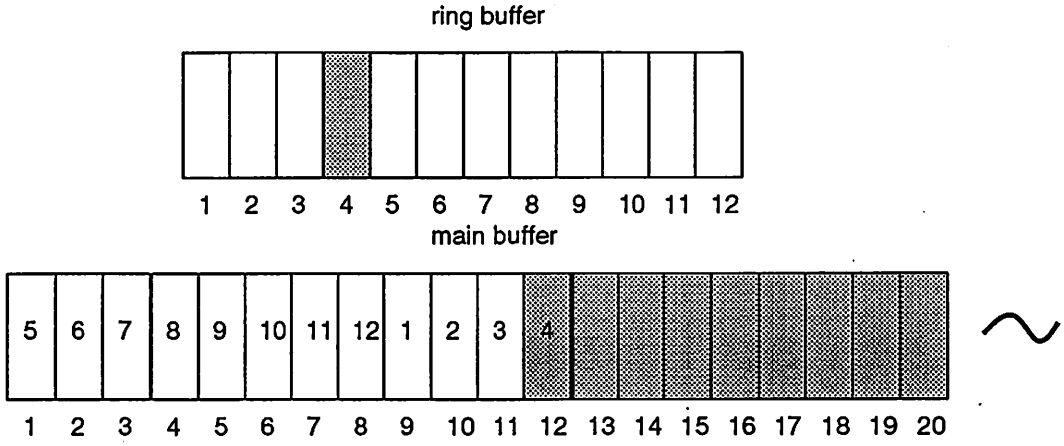

Figure 4.1: Slot arrangment for ring and main buffer

To ensure real time performance, only every Nth value is used for the calculation of the absolute mean of each slot ($N = 4$ for HP, $N = 10$ for DEC and SUN). For example, on SUN the absolute mean for a slot is calculated from $1200/10 = 120$ data values.

4.2 Fine Phrase Segmentation

A set of power and zero-crossing thresholds facilitates fine segmentation of the utterance into phrases. Fig. 4.2 depicts the most important of these thresholds. Power thresholds are not absolute but relative to minimum and maximum values during the current utterance. For details refer to the source code in `segment.c` and `newsegment.c` (function `segment_phrase()`) and see also [2][1].

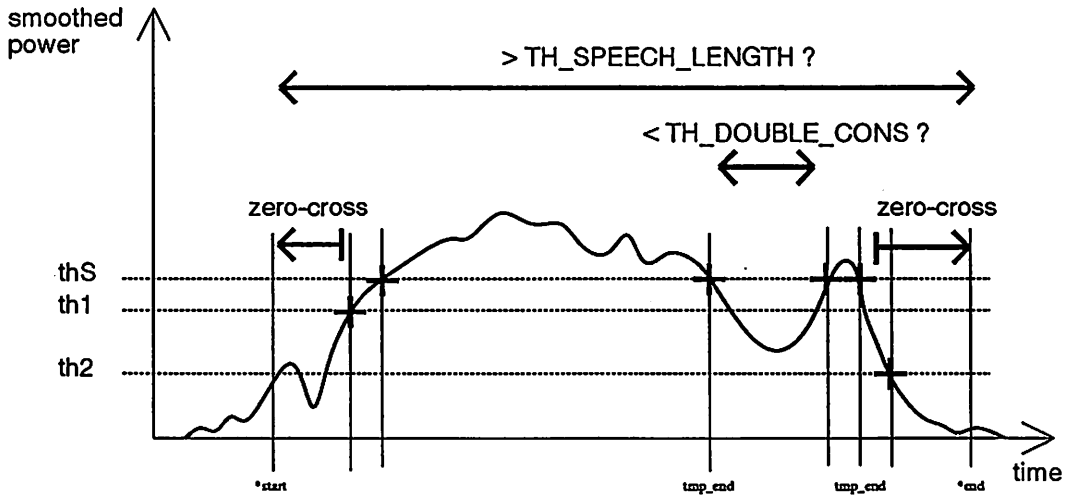

Figure 4.2: Threshold logic for fine segmentation

Chapter 5

Installation

5.1 Files and Libraries

SpeechIn and SpeechPress were installed on several machine architectures, several operating systems and different DASBOXs. Table 5.1 shows the current list of implementations. For each implementation exist two makefiles:

makeXXX	SpeechIn
makeXXX_P	SpeechPress

XXX stands for any of the abbreviations in Table 5.1, i.e. the makefile for SpeechPress on DECstation with ULTRIX 4.2a would be called makeDEC4.2a_P.

The necessary source files differ for the various implementations. Table 5.2 shows, which source files are needed for which machine configuration. All source files are managed with the revision control system RCS.

Additionally, several shell scripts shown in Table 5.3 provide support for compilation etc..

Table 5.1: Machines and operating systems

abbreviation	*description*	*devicename*	*bus*
DEC	DECstation 5000/200, ULTRIX 4.0/4.1	/dev/scsidasbox	SCSI
DEC4.2a	DECstation 5000/200, ULTRIX 4.2a	/dev/dasbox	SCSI
DECTB	DECstation 5000/200, ULTRIX 4.2a	/dev/dm0	TURBOchannel
ALPHA	DEC Alpha station	/dev/dasbox	SCSI
HP	HP 9000/7xx, HP-UX 8.05, DASBOX16	/dev/das0	AT
HPOLD	HP 9000/7xx , HP-UX 8.05	/dev/dm0	AT
SUN	Sun SPARC, SunOS 4.1.1	/dev/dm0	VME
SUNS	Sun SPARC, SunOS 4.1.1	/dev/dm0	S

Table 5.2: Source files

program and description	DEC	DEC4.2a	DECTB	ALPHA	HP	HPOLD	SUN	SUNS
`segment.c` main (SpeechIn)	X	X	X	X	X	X	X	X
`newsegment.c` main (SpeechPress)	X	X	X	X	X	X	X	X
`header.h` dataformat for stdout	X	X	X	X	X	X	X	X
`XG.c` graphic subroutines	X	X	X	X	X	X	X	X
`time.c` measuring cpu-time	X	X	X	X	X	X	X	X
`swap.c` byte swapping	X	X	X	X	X	X	X	X
`adin.c` high-level AD	X	X	X	X		X	X	X
`daout.c` high-level DA	X	X	X	X		X	X	X
`adin_new.c` high-level AD					X			
`daout_new.c` high-level DA					X			
`dasbox.h` DASBOX header			X			X	X	X
`d_hp.c` DASBOX interface						X		
`dmioctl.h` AT-DMAC(SDS-9117)						X		
`das90.h` DASBOX16 header					X			
`d_hpnew.c` DASBOX16 interface					X			
`dasioctl.h` AT-DMAC(SDS-9117)					X			
`d_sun.c` DASBOX interface							X	
`udm.h` VME-DMAC(SDS-8600)							X	
`d_tb.c` DASBOX interface			X					
`udm_tb.h` TurboDRC(SDS-9035)			X					
`d_suns.c` DASBOX interface								X
`udm_s.h` S-DMAC(SDS-9004)								X
`hand_signal.c` interrupt handling	X	X	X	X				
`xaif.h` xa driver	X	X	X	X				
`dasdef_con.h` DASBOX header	X	X		X				
`/usr/dasbox/support/dasbox.a` library	X		X					
`/usr/dasbox/gsc/_lib/libgsc.a` library	X		X					
`/usr/dasbox/src/dasbox.a` library		X		X				
`/usr/dasbox/lib/lib_uagt.a` library		X		X				

Table 5.3: Script files

filename	description
`compile_all.csh`	rsh and make
`install.csh`	copying only necessary files for each implementation
`test_all.csh`	tests simultaneously some machines (not up-to-date)
`/usr/common/bin/SpeechIn`	machine independent script
`/usr/common/bin/SpeechPress`	machine independent script
`demonstration.csh`	demo of SpeechIn (not up-to-date)

5.2 Compilation

Simply run the shell script `compile_all.csh`. Make sure that you can run `rsh` on the relevant machines. For details see `man rsh` and the source code of `compile_all.csh`. If you just want to compile for one machine enter the following commands (e.g. for HP):

```
$ login atrp13 -l demoHP
$ cd /NFS/atr-fs/pub1/common/SpeechIn
$ make -f makeHP all          # for SpeechIn
$ make -f makeHP_P all        # for SpeechPress
```

which compiles executables into `/NFS/atr-fs/pub1/common/SpeechIn/BINHP`.

5.3 Updates

When you perform any changes , e.g. bugfixes, please use RCS commands for revision management.

```
$ cd /NFS/atr-fs/pub1/common/SpeechIn/SRC
$ co -l newsegment.c
  (edit file)
$ cd ..
$ make -f makeHP all
  (verify that program does what it should do)
$ ci -u newsegment.c
  (write comment)
$ compile_all.csh
```

You can verify which version is running by using `ident`.

```
$ ident SpeechPress
SpeechPress:
     $Header: crt0.s,v 66.10 91/02/25 18:10:31 ssa Rel $
     $Header: mapdld.c,v 66.14 90/10/29 18:22:28 shoe Rel $
     $Header: RCS/newsegment.c,v 1.11 1993/03/01 05:48:31 singer Rel $
     $Header: RCS/XG.c,v 1.1 1992/10/23 13:50:56 singer Rel singer $
     $Header: RCS/adin_new.c,v 1.1 92/10/23 13:50:59 singer Rel $
     $Header: RCS/d_hpnew.c,v 1.1 1992/10/23 13:51:01 singer Rel $
     $Header: RCS/daout_new.c,v 1.1 1992/10/23 13:51:09 singer Rel singer $
     $Header: RCS/swap.c,v 1.3 1992/11/16 12:54:33 singer Rel $
     $Header: RCS/time.c,v 1.2 1992/10/23 14:16:30 singer Rel $
```

The current version of `segment.c` and `newsegment.c` is usually displayed in the upper right hand side of the wave window. If you want changes in other files reflected on the screen, check-out and check-in `segment.c` and `newsegment.c` to update the revision number.

The `rcs` state of the released version has been set to `Rel` on March 9th, 1993.

Acknowledgements

The authors are grateful to Dr. Kurematsu, the president of ATR Interpreting Telephony Research Laboratories, and for the support and help received from all colleagues at ATR Interpreting Telephony Research Laboratories.

We are especially grateful to Dr. Kawabata (NTT Human Interface), H. Hattori (NEC) and K. Ohkura (ATR), who wrote the predecessor programs and support from T. Ban (SET) and K. Takashima (SET).

Bibliography

[1] L. Lamel, L. Rabiner, A. Rosenberg, and J. Wilpon. An improved endpoint detector for isolated word recognition. *Transactions on Acoustics, Speech, and Signal Processing*, 29(4):777–785, 1981.

[2] L.R. Rabiner and M. Sambur. An algorithm for determining the endpoints of isolated utterances. *The Bell System Technical Journal*, 54:297–315, Feb 1975.

[3] M. Sugiyama. Computer & software user's guide in ATR ITL/ speech processing department. Technical Report TR-I-0300, ATR, 1993. (in Japanese).