

```

1
2
3     typedef double XFLOAT
4     typedef double OTA_FLOAT
5
6     typedef double OTA_FLOAT
7     typedef MAT_DCplx OTA_CPLX
8
9
10 {
11
12 typedef struct
13 {
14     float FrameWeightWeight
15     bool   UseRelDistance
16     float ViterbiDistanceWeightFactor
17 } VITERBI_PARA
18
19 typedef struct
20 {
21     long Samplerate
22     int  mSRDetectFineAlignCorrlen
23     int  mDelayFineAlignCorrlen
24     int  WindowSize[8]
25     int  CoarseAlignCorrlen[8]
26     float pViterbiDistanceWeightFactor[8]
27 } SPEECH_WINDOW_PARA
28
29 typedef struct
30 {
31     SPEECH_WINDOW_PARA Win[3]
32     float LowEnergyThresholdFactor
33     float LowCorrelThreshold
34
35     float FineAlignLowEnergyThresh
36     float FineAlignLowEnergyCorrel
37     float FineAlignShortDropOfCorrelR
38     float FineAlignShortDropOfCorrelRLastBest
39     float ViterbiDistanceWeightFactorDist
40     float ViterbiDistanceWeightFactor
41 } SPEECH_TA_PARA
42
43 typedef struct
44 {
45     SPEECH_WINDOW_PARA Win[3]
46     float LowEnergyThresholdFactor
47     float LowCorrelThreshold
48
49     float FineAlignLowEnergyThresh
50     float FineAlignLowEnergyCorrel
51     float FineAlignShortDropOfCorrelR
52     float FineAlignShortDropOfCorrelRLastBest
53     float ViterbiDistanceWeightFactorDist
54     float ViterbiDistanceWeightFactor
55 } AUDIO_TA_PARA
56
57 typedef struct
58 {
59     float mCorrForSkippingInitialDelaySearch
60     int  CoarseAlignSegmentLengthInMs
61 } GENERAL_TA_PARA
62
63 typedef struct
64 {
65     void Init(long Samplerate)
66     {
67         if (Samplerate==16000)    MaxWin=4
68         else if (Samplerate==8000) MaxWin=4

```

```

69         else                                     MaxWin=4
70
71         LowPeakEliminationThreshold= 0.2000000029802322
72
73         if (Samplerate==16000)      PercentageRequired = 0.05F
74         else if (Samplerate==8000)  PercentageRequired = 0.1F
75         else                        PercentageRequired = 0.02F
76
77         MaxDistance = 14
78
79         MinReliability = 7
80
81         PercentageRequired = 0.7
82         OTA_FLOAT MaxGradient = 1.1
83         OTA_FLOAT MaxTimescaling = 0.1
84
85         if (Samplerate==48000)      MaxStepPerFrame = MaxGradient * 1024.0
86         else if (Samplerate==8000)  MaxStepPerFrame = MaxGradient * 128.0
87         MaxBins = ((int)(MaxStepPerFrame*2.0*0.9))
88         MaxStepPerFrame *= 4
89
90     }
91
92     float LowEnergyThresholdFactor
93     float LowCorrelThreshold
94
95     int     MaxStepPerFrame
96     int     MaxBins
97     int     MaxWin
98     int     MinHistogramData
99
100    float   MinReliability
101
102    double  LowPeakEliminationThreshold
103    float   MinFrequencyOfOccurrence
104    float   LargeStepLimit
105
106    float   MaxDistanceToLast
107    float   MaxDistance
108    float   MaxLargeStep
109
110    float   ReliabilityThreshold
111    float   PercentageRequired
112
113    float   AllowedDistancePara2
114    float   AllowedDistancePara3
115 } SR_ESTIMATION_PARA
116
117 class CParameters
118 {
119     public:
120         CParameters()
121         {
122             int i
123             mTAPara.mCorrForSkippingInitialDelaySearch = 0.6F
124             mTAPara.CoarseAlignSegmentLengthInMs = 600
125
126             SPEECH_WINDOW_PARA      SpeechWinPara[] =
127             {
128                 {8000, 32, 32,
129                  {128, 256, 128, 64, 32, 0, 0},
130                  {-1, -1, -1, 86, 34, 0, 0},
131                  {-1, -1, -1, 15, 12, 0, 0}},
132                 {16000, 64, 64,
133                  {256, 512, 256, 128, 64, 0},
134                  {-1, -1, -1, 63, 33, 0},
135                  {-1, -1, -1, 13, 10, 0}},
136                 {48000, 256, 256,

```

```

137         {512, 1024, 512, 512, 128, 0},
138         {-1, -1, -1, 115, 61, 0},
139         {-1, -1, -1, 17, 16, 0}}
140     }
141
142     for (i=0 i<3 i++)
143     {
144         mSpeechTAPara.Win[i].Samplerate = SpeechWinPara[i].Samplerate
145         mSpeechTAPara.Win[i].mDelayFineAlignCorrlen = SpeechWinPara[i].mDelayFineAlignCorrlen
146         mSpeechTAPara.Win[i].mSRDetectFineAlignCorrlen =
SpeechWinPara[i].mSRDetectFineAlignCorrlen
147         for (int k=0 k<8 k++)
148         {
149             mSpeechTAPara.Win[i].CoarseAlignCorrlen[k] =
SpeechWinPara[i].CoarseAlignCorrlen[k]
150             mSpeechTAPara.Win[i].WindowSize[k] = SpeechWinPara[i].WindowSize[k]
151             mSpeechTAPara.Win[i].pViterbiDistanceWeightFactor[k] =
SpeechWinPara[i].pViterbiDistanceWeightFactor[k]
152         }
153     }
154     mSpeechTAPara.LowEnergyThresholdFactor = 15.0F
155     mSpeechTAPara.LowCorrelThreshold = 0.4F
156     mSpeechTAPara.FineAlignLowEnergyThresh = 2.0
157     mSpeechTAPara.FineAlignLowEnergyCorrel = 0.6F
158     mSpeechTAPara.FineAlignShortDropOfCorrelR = -1
159     mSpeechTAPara.FineAlignShortDropOfCorrelRLastBest = 0.65F
160
161     mSpeechTAPara.ViterbiDistanceWeightFactorDist = 5
162
163     SPEECH_WINDOW_PARA AudioWinPara[] =
164     {
165         {8000, 32, 32,
166          {64, 128, 64, 64, 16, 0, 0},
167          {-1, -1, -1, 128, 32, 0, 0},
168          {-1, -1, -1, 6, 6, 0, 0}},
169         {16000, 64, 64,
170          {128, 256, 128, 128, 32, 0},
171          {-1, -1, -1, 64, 32, 0},
172          {-1, -1, -1, 12, 12, 0}},
173         {48000, 256, 2048,
174          {512, 1024, 512, 512, 256, 128, 0},
175          {-1, -1, -1, 512, 1024, 2048, 0},
176          {-1, -1, -1, 16, 16, 32, 0}}
177     }
178
179     for (i=0 i<3 i++)
180     {
181         mAudioTAPara.Win[i].Samplerate = AudioWinPara[i].Samplerate
182         mAudioTAPara.Win[i].mDelayFineAlignCorrlen =
AudioWinPara[i].mDelayFineAlignCorrlen
183         mAudioTAPara.Win[i].mSRDetectFineAlignCorrlen =
AudioWinPara[i].mSRDetectFineAlignCorrlen
184         for (int k=0 k<8 k++)
185         {
186             mAudioTAPara.Win[i].CoarseAlignCorrlen[k] = AudioWinPara[i].CoarseAlignCorrlen[k]
187             mAudioTAPara.Win[i].WindowSize[k] = AudioWinPara[i].WindowSize[k]
188             mAudioTAPara.Win[i].pViterbiDistanceWeightFactor[k] =
AudioWinPara[i].pViterbiDistanceWeightFactor[k]
189         }
190     }
191     mAudioTAPara.LowEnergyThresholdFactor = 1
192     mAudioTAPara.LowCorrelThreshold = 0.85F
193     mAudioTAPara.FineAlignLowEnergyThresh = 32.0
194     mAudioTAPara.FineAlignLowEnergyCorrel = 0.8F
195     mAudioTAPara.FineAlignShortDropOfCorrelR = -1
196     mAudioTAPara.FineAlignShortDropOfCorrelRLastBest = 0.8F

```

```

197         mAudioTAPara.ViterbiDistanceWeightFactorDist = 6
198
199         mSREPara.LowEnergyThresholdFactor = 15.0F
200         mSREPara.LowCorrelThreshold = 0.4F
201
202         mSREPara.MaxStepPerFrame = 160
203         mSREPara.MaxBins = ((int)(mSREPara.MaxStepPerFrame*2.0*0.9))
204
205         mSREPara.MaxWin=4
206         mSREPara.LowPeakEliminationThreshold=0.2000000029802322F
207         mSREPara.PercentageRequired = 0.04F
208
209         mSREPara.LargeStepLimit = 0.08F
210         mSREPara.MaxDistanceToLast = 7
211         mSREPara.MaxLargeStep = 5
212         mSREPara.MaxDistance = 14
213
214         mSREPara.MinReliability = 7
215         mSREPara.MinFrequencyOfOccurrence = 3
216
217         mSREPara.AllowedDistancePara2 = 0.85F
218         mSREPara.AllowedDistancePara3 = 1.5F
219
220         mSREPara.ReliabilityThreshold = 0.3F
221         mSREPara.MinHistogramData = 8
222
223         mViterbi.UseRelDistance = false
224         mViterbi.FrameWeightWeight = 1.0F
225     }
226
227     void Init(long Samplerate)
228     {
229         mSREPara.Init(Samplerate)
230     }
231
232     VITERBI_PARA      mViterbi
233     GENERAL_TA_PARA   mTAPara
234     SPEECH_TA_PARA    mSpeechTAPara
235     AUDIO_TA_PARA     mAudioTAPara
236     SR_ESTIMATION_PARA mSREPara
237 }
238 }
239
240
241 {
242
243 class CProcessData
244 {
245     public:
246         CProcessData()
247         {
248             int i
249
250             mCurrentIteration = -1
251             mStartPlotIteration=10
252             mLastPlotIteration =10
253             mEnablePlotting=false
254             mpLogFile = 0
255
256             mWindowSize = 2048
257             mSRDetectFineAlignCorrlen = 1024
258             mDelayFineAlignCorrlen = 1024
259             mOverlap      = 1024
260             mSamplerate = 48000
261             mNumSignals = 0
262             mpMathlibHandle = 0
263             mMinLowVarDelay = -99999999
264             mMaxHighVarDelay = 99999999

```

```

265
266     mMinStaticDelayInMs = -2500
267     mMaxStaticDelayInMs = 2500
268
269     mMaxToleratedRelativeSamplerateDifference = 1.0
270
271     for (i=0 i<8 i++)
272         mpViterbiDistanceWeightFactor[i] = 0.0001F
273 }
274
275 int mMinStaticDelayInMs
276 int mMaxStaticDelayInMs
277
278 int mMinLowVarDelayInSamples
279 int mMaxHighVarDelayInSamples
280
281 int mStartPlotIteration
282 int mLastPlotIteration
283 bool mEnablePlotting
284 long mSamplerate
285
286 FILE* mpLogFile
287
288 int mCurrentIteration
289
290 int mpWindowSize[8]
291
292 int mpOverlap[8]
293
294 int mpCoarseAlignCorrlen[8]
295
296 float mpViterbiDistanceWeightFactor[8]
297
298 int mDelayFineAlignCorrlen
299 int mSRDetectFineAlignCorrlen
300 float mMaxToleratedRelativeSamplerateDifference
301 int mWindowSize
302
303 int mOverlap
304
305 int mCoarseAlignCorrlen
306
307 int mNumSignals
308 void* mpMathlibHandle
309
310 int mMinLowVarDelay
311 int mMaxHighVarDelay
312 int mStepSize
313
314 bool Init(int Iteration, float MoreDownsampling)
315 {
316     assert(MoreDownsampling)
317
318     mCurrentIteration = Iteration
319     mP.Init(mSamplerate)
320
321     mWindowSize = (int)((float)mpWindowSize[Iteration]*MoreDownsampling)
322     mOverlap = (int)((float)mpOverlap[Iteration]*MoreDownsampling)
323     mCoarseAlignCorrlen = mpCoarseAlignCorrlen[Iteration]
324     mStepSize = mWindowSize - mOverlap
325     mMinLowVarDelay = mMinLowVarDelayInSamples / mStepSize
326     mMaxHighVarDelay = mMaxHighVarDelayInSamples / mStepSize
327
328     float D = mpViterbiDistanceWeightFactor[Iteration]
329     D = D * mSamplerate / mStepSize / 1000
330     float F = ((float)log(1+0.5)) / (D*D)
331     mP.mViterbi.ViterbiDistanceWeightFactor = F
332

```

```

333         D = mP.mSpeechTAPara.ViterbiDistanceWeightFactorDist
334         D = D * mSamplerate / 1000
335         F = ((float) log(1+0.5) / (D*D))
336         mP.mSpeechTAPara.ViterbiDistanceWeightFactor = F
337
338         return true
339     }
340
341     CParameters    mP
342 }
343
344 class SECTION
345 {
346     public:
347         int Start
348         int End
349         int Len() {return End-Start }
350         void CopyFrom(const SECTION &src)
351         {
352             this->Start = src.Start
353             this->End    = src.End
354         }
355     }
356
357 typedef struct OTA_RESULT
358 {
359     void CopyFrom(const OTA_RESULT* src)
360     {
361         mNumFrames      = src->mNumFrames
362         mStepsize        = src->mStepsize
363         mResolutionInSamples = src->mResolutionInSamples
364         if (src->mpDelay != NULL && mNumFrames > 0)
365         {
366             matFree(mpDelay)
367             mpDelay = (long*)matMalloc(mNumFrames * sizeof(long))
368             for (int i = 0 i < mNumFrames i++)
369                 mpDelay[i] = src->mpDelay[i]
370         }
371         else
372         {
373             matFree(mpDelay)
374             mpDelay = NULL
375         }
376
377         if (src->mpReliability != NULL && mNumFrames > 0)
378         {
379             matFree(mpReliability)
380             mpReliability = (OTA_FLOAT*)matMalloc(mNumFrames * sizeof(OTA_FLOAT))
381             for (int i = 0 i < mNumFrames i++)
382                 mpReliability[i] = src->mpReliability[i]
383         }
384         else
385         {
386             matFree(mpReliability)
387             mpReliability = NULL
388         }
389         mAvgReliability = src->mAvgReliability
390         mRelSamplerateDev = src->mRelSamplerateDev
391
392         mNumUtterances = src->mNumUtterances
393         if (src->mpStartSampleUtterance != NULL && mNumUtterances > 0)
394         {
395             matFree(mpStartSampleUtterance)
396             mpStartSampleUtterance = (int*)matMalloc(mNumUtterances * sizeof(int))
397             for (int i = 0 i < mNumUtterances i++)
398                 mpStartSampleUtterance[i] = src->mpStartSampleUtterance[i]
399         }
400         else

```

```

401     {
402         matFree(mpStartSampleUtterance)
403         mpStartSampleUtterance = NULL
404     }
405     if (src->mpStopSampleUtterance != NULL && mNumUtterances > 0)
406     {
407         matFree(mpStopSampleUtterance)
408         mpStopSampleUtterance = (int*)matMalloc(mNumUtterances * sizeof(int))
409         for (int i = 0 i < mNumUtterances i++)
410             mpStopSampleUtterance[i] = src->mpStopSampleUtterance[i]
411     }
412     else
413     {
414         matFree(mpStopSampleUtterance)
415         mpStopSampleUtterance = NULL
416     }
417     if (src->mpDelayUtterance != NULL && mNumUtterances > 0)
418     {
419         matFree(mpDelayUtterance)
420         mpDelayUtterance = (int*)matMalloc(mNumUtterances * sizeof(int))
421         for (int i = 0 i < mNumUtterances i++)
422             mpDelayUtterance[i] = src->mpDelayUtterance[i]
423     }
424     else
425     {
426         matFree(mpDelayUtterance)
427         mpDelayUtterance = NULL
428     }
429
430     mNumSections = src->mNumSections
431     if (src->mpRefSections != NULL && mNumSections > 0)
432     {
433         delete[] mpRefSections
434         mpRefSections = new SECTION[mNumSections]
435         for (int i = 0 i < mNumSections i++)
436             mpRefSections[i].CopyFrom(src->mpRefSections[i])
437     }
438     else
439     {
440         delete[] mpRefSections
441         mpRefSections = NULL
442     }
443     if (src->mpDegSections != NULL && mNumSections > 0)
444     {
445         delete[] mpDegSections
446         mpDegSections = new SECTION[mNumSections]
447         for (int i = 0 i < mNumSections i++)
448             mpDegSections[i].CopyFrom(src->mpDegSections[i])
449     }
450     else
451     {
452         delete[] mpDegSections
453         mpDegSections = NULL
454     }
455
456     mSNRRefdB = src->mSNRRefdB
457     mSNRDegdB = src->mSNRDegdB
458     mNoiseLevelRef = src->mNoiseLevelRef
459     mNoiseLevelDeg = src->mNoiseLevelDeg
460     mSignalLevelRef = src->mSignalLevelRef
461     mSignalLevelDeg = src->mSignalLevelDeg
462     mNoiseThresholdRef = src->mNoiseThresholdRef
463     mNoiseThresholdDeg = src->mNoiseThresholdDeg
464
465     if (src->mpActiveFrameFlags != NULL && mNumFrames > 0)
466     {
467         matFree(mpActiveFrameFlags)
468         mpActiveFrameFlags = (int*)matMalloc(mNumFrames * sizeof(int))

```

```

469         for (int i = 0 i < mNumFrames i++)
470             mpActiveFrameFlags[i] = src->mpActiveFrameFlags[i]
471     }
472     else
473     {
474         matFree(mpActiveFrameFlags)
475         mpActiveFrameFlags = NULL
476     }
477
478     if (src->mpIgnoreFlags != NULL && mNumFrames > 0)
479     {
480
481         matFree(mpIgnoreFlags)
482         mpIgnoreFlags = (int*)matMalloc(mNumFrames * sizeof(int))
483         mNumIngoreFlags = src->mNumIngoreFlags
484         for (int i = 0 i < mNumFrames i++)
485             mpIgnoreFlags[i] = src->mpIgnoreFlags[i]
486     }
487     else
488     {
489         matFree(mpIgnoreFlags)
490         mpIgnoreFlags = NULL
491     }
492
493     for (int i = 0 i < 5 i++)
494         mTimeDiffs[i] = src->mTimeDiffs[i]
495
496     mAslFrames = src->mAslFrames
497     mAslFramelength = src->mAslFramelength
498     if (src->mpAslActiveFrameFlags != NULL && mAslFrames > 0)
499     {
500         matFree(mpAslActiveFrameFlags)
501         mpAslActiveFrameFlags = (int*)matMalloc(mAslFrames * sizeof(int))
502         for (int i = 0 i < mAslFrames i++)
503             mpAslActiveFrameFlags[i] = src->mpAslActiveFrameFlags[i]
504     }
505     else
506     {
507         matFree(mpAslActiveFrameFlags)
508         mpAslActiveFrameFlags = NULL
509     }
510
511     mAslFramesDeg = src->mAslFramesDeg
512     if (src->mpAslActiveFrameFlagsDeg != NULL && mAslFramesDeg > 0)
513     {
514         matFree(mpAslActiveFrameFlagsDeg)
515         mpAslActiveFrameFlagsDeg = (int*)matMalloc(mAslFramesDeg * sizeof(int))
516         for (int i = 0 i < mAslFramesDeg i++)
517             mpAslActiveFrameFlagsDeg[i] = src->mpAslActiveFrameFlagsDeg[i]
518     }
519     else
520     {
521         matFree(mpAslActiveFrameFlagsDeg)
522         mpAslActiveFrameFlagsDeg = NULL
523     }
524
525     FirstRefSample = src->FirstRefSample
526     FirstDegSample = src->FirstDegSample
527 }
528
529 OTA_RESULT()
530 {
531     mNumFrames = 0
532     mpDelay = NULL
533
534     mpReliability = NULL
535
536     mNumUtterances = 0

```



```

537     mpStartSampleUtterance = NULL
538     mpStopSampleUtterance  = NULL
539     mpDelayUtterance       = NULL
540
541     mNumSections = 0
542     mpRefSections = NULL
543     mpDegSections = NULL
544
545     mpActiveFrameFlags = NULL
546     mpIgnoreFlags = NULL
547     mNumIngoreFlags = 0
548
549     mAslFramelength = 0
550     mAslFrames = 0
551     mpAslActiveFrameFlags = NULL
552     mAslFramesDeg = 0
553     mpAslActiveFrameFlagsDeg = NULL
554
555     FirstRefSample = FirstDegSample = 0
556 }
557
558 ~OTA_RESULT()
559 {
560     matFree(mpDelay)
561     mpDelay = NULL
562
563     matFree(mpReliability)
564     mpReliability = NULL
565
566     matFree(mpStartSampleUtterance)
567     mpStartSampleUtterance = NULL
568
569     matFree(mpStopSampleUtterance)
570     mpStopSampleUtterance = NULL
571
572     matFree(mpDelayUtterance)
573     mpDelayUtterance = NULL
574
575     delete[] mpRefSections
576     mpRefSections = NULL
577     delete[] mpDegSections
578     mpDegSections = NULL
579
580     matFree(mpActiveFrameFlags)
581     mpActiveFrameFlags = NULL
582
583     matFree(mpIgnoreFlags)
584     mpIgnoreFlags = NULL
585
586     matFree(mpAslActiveFrameFlags)
587     mpAslActiveFrameFlags = NULL
588     matFree(mpAslActiveFrameFlagsDeg)
589     mpAslActiveFrameFlagsDeg = NULL
590 }
591
592 long mNumFrames
593 int mStepsize
594 int mResolutionInSamples
595 int mPitchFrameSize
596 long *mpDelay
597 OTA_FLOAT *mpReliability
598 OTA_FLOAT mAvgReliability
599 OTA_FLOAT mRelSamplerateDev
600
601 int mNumUtterances
602 int* mpStartSampleUtterance
603 int* mpStopSampleUtterance
604 int* mpDelayUtterance

```

```

605     int FirstRefSample
606     int FirstDegSample
607
608     int          mNumSections
609     SECTION      *mpRefSections
610     SECTION      *mpDegSections
611
612     double mSNRRefdB, mSNRDegdB
613     double mNoiseLevelRef, mNoiseLevelDeg
614     double mSignalLevelRef, mSignalLevelDeg
615     double mNoiseThresholdRef, mNoiseThresholdDeg
616
617     int *mpActiveFrameFlags
618
619     int *mpIgnoreFlags
620     int mNumIgnoreFlags
621     int mAslFrames
622     int mAslFrameLength
623     int *mpAslActiveFrameFlags
624     int mAslFramesDeg
625     int *mpAslActiveFrameFlagsDeg
626
627     double mTimeDiffs[5]
628
629 }OTA_RESULT
630
631 struct FilteringParameters
632 {
633     int pListeningCondition
634     double cutOffFrequencyLow
635     double cutOffFrequencyHigh
636     double disturbedEnergyQuotient
637 }
638
639 class ITempAlignment
640 {
641     public:
642
643     virtual bool Init(CProcessData* pProcessData)=0
644     virtual void Free()=0
645     virtual void Destroy()=0
646
647     virtual bool SetSignal(int Index, unsigned long SampleRate, unsigned long NumSamples, int
NumChannels, OTA_FLOAT** pSignal)=0
648
649     virtual void GetFilterCharacteristics(FilteringParameters *FilterParams)=0
650
651     virtual bool FilterSignal(int Index, FilteringParameters *FilterParams)=0
652
653     virtual bool Run(unsigned long Control, OTA_RESULT* pResult, int TArunIndex)=0
654
655     virtual void GetNoiseSwitching(OTA_FLOAT* pBGNSwitchingLevel, OTA_FLOAT*
pNoiseLevelSpeechDeg, OTA_FLOAT* pNoiseLevelSilenceDeg)=0
656
657     virtual OTA_FLOAT GetPitchFreq(int Signal, int Channel)=0
658
659     virtual OTA_FLOAT GetPitchVector(int Signal, int Channel, OTA_FLOAT* pVector, int NumFrames,
int SamplesPerFrame)=0
660     virtual int GetPitchFrameSize()=0
661 }
662
663 enum AlignmentType
664 {
665     TA_FOR_SPEECH=0,
666
667 }
668
669 ITempAlignment* CreateAlignment(AlignmentType Type)

```

```

670
671 }
672
673
674 {
675
676 class CEnergyFeature : public CFeature
677 {
678     public:
679         CEnergyFeature()
680         {
681             for (int i=0 i<2*2*2 i++)
682                 mpFeatureVectors[i] = 0
683             mNumFVectors = 0
684         }
685
686         virtual ~CEnergyFeature() {Free() }
687         void Free()
688         virtual bool CalcVector(CTASignal** pSignals, CProcessData* pProcessData, SEGMENT* pSegments,
int* pActiveFrameFlags)
689         virtual CFeatureVector* GetFVector(int IndexOfVector)
690         virtual int GetNumSets() {return 2 }
691
692     private:
693         OTA_FLOAT AverageEnergy(OTA_FLOAT* data, unsigned long length)
694         OTA_FLOAT AverageEnergyHannWindowed(OTA_FLOAT* data, unsigned long length)
695         bool CalcFeatureForOneChannel(int FVecIndex, CAudioSignal* Sig, int Channel, SEGMENT*
pSegment, int* pActiveFrameFlags)
696         CFeatureVector *mpFeatureVectors[2*2*2]
697         CProcessData mProcessData
698
699         unsigned int mStepSize
700         unsigned long mpFLength[2*2*2]
701         int mNumFVectors
702 }
703
704 void CEnergyFeature::Free()
705 {
706     for (int i = 0 i < 2*2*2 i++)
707     {
708         delete mpFeatureVectors[i]
709         mpFeatureVectors[i] = NULL
710         mpFLength [i] = 0
711     }
712     mNumFVectors = 0
713     mChannels = 0
714 }
715
716 CFeature* CreateEnergyFeature()
717 {
718     return (CFeature*) new CEnergyFeature
719 }
720
721 OTA_FLOAT CEnergyFeature::AverageEnergyHannWindowed(OTA_FLOAT* Data, unsigned long Length)
722 {
723     OTA_FLOAT Energy
724     OTA_FLOAT *pf = matxMalloc(Length)
725     matWinHann(Data, pf, Length)
726     Energy = matDotProd(pf, pf, Length) / Length
727     matFree(pf)
728     return Energy
729 }
730
731 OTA_FLOAT CEnergyFeature::AverageEnergy(OTA_FLOAT* Data, unsigned long Length)
732 {
733     OTA_FLOAT Energy
734
735     Energy = matDotProd(Data, Data, Length) / Length

```

```

736     return Energy
737 }
738
739 void HighpassFilter(OTA_FLOAT* pData, int NumSamples, OTA_FLOAT NormalizedCutOffFreq, MAT_HANDLE mh)
740 {
741
742     OTA_FLOAT pTaps[128]
743
744     matGenHighPassCoefficients(NormalizedCutOffFreq, pTaps, 128, MAT_WinHann)
745     OTA_FLOAT* pSig = (OTA_FLOAT*)matMalloc(NumSamples * sizeof(OTA_FLOAT))
746
747     int i
748     matRunFIRFilter(mh, pData, pSig, NumSamples, pTaps, 128, MAT_FIRNoDelayComp)
749     matbCopy(pSig+128, pData, NumSamples-128)
750     for (i=NumSamples-128 i<NumSamples i++)
751         pData[i] = 0
752
753     matFree(pSig)
754
755 }
756
757 bool CEnergyFeature::CalcFeatureForOneChannel(int FVecIndex, CAudioSignal* Sig, int Channel, SEGMENT*
pSegment, int* pActiveFrameFlags)
758 {
759     bool rc = true
760
761     unsigned long StartSample=pSegment->Start
762     unsigned long FrameCount=0
763
764     OTA_FLOAT* pSamples = Sig->mpData[Channel]
765
766     int Framesize = mProcessData.mStepSize
767     int Overlap = 0
768     unsigned long LastStart=((pSegment->End-StartSample-Framesize+1) / mStepSize) *
mStepSize+StartSample
769
770     OPTTRY
771     {
772
773         mpFLength[FVecIndex] = (pSegment->End-StartSample-Framesize+1) / mStepSize
774         mpFeatureVectors[FVecIndex] = new CFeatureVector(mpFLength[FVecIndex])
775         mpFeatureVectors[FVecIndex+2*2] = new CFeatureVector(mpFLength[FVecIndex])
776         mpFLength[FVecIndex+2*2] = mpFLength[FVecIndex]
777
778     }
779     OPTCATCH(...)
780     {
781         int x=1
782         DebugBreak()
783     }
784
785     OPTTRY
786     {
787         if(pActiveFrameFlags)
788         {
789             FrameCount=0
790             while (StartSample<LastStart)
791             {
792                 if (pActiveFrameFlags[FrameCount])
793                     mpFeatureVectors[FVecIndex]->mpVector[FrameCount] =
AverageEnergy(&pSamples[StartSample], Framesize)
794                 else
795                     mpFeatureVectors[FVecIndex]->mpVector[FrameCount] = 0
796                 StartSample += mStepSize
797                 FrameCount++
798             }
799         }
800         else

```

```

801     {
802         FrameCount=0
803         while (StartSample<LastStart)
804         {
805             mpFeatureVectors[FVecIndex]->mpVector[FrameCount] =
AverageEnergy(&pSamples[StartSample], Framesize)
806             StartSample += mStepSize
807             FrameCount++
808         }
809     }
810 }
811 OPTCATCH((...))
812 {
813     int x=1
814     DebugBreak()
815 }
816 OPTTRY
817 {
818     OTA_FLOAT* pDest = mpFeatureVectors[FVecIndex+2*2]->mpVector
819     matbCopy(mpFeatureVectors[FVecIndex]->mpVector, pDest, FrameCount)
820     matbAdd1(1, pDest, FrameCount)
821
822     for (unsigned int i=0 i<FrameCount i++)
823         pDest[i] = pow(pDest[i], 0.25)
824
825 }
826 OPTCATCH((...))
827 {
828     int x=1
829     DebugBreak()
830 }
831
832 mNumFVectors++
833
834 return rc
835 }
836
837 bool CEnergyFeature::CalcVector(CTASignal** pSignalsIn, CProcessData* pProcessData, SEGMENT*
pSegments, int* pActiveFrameFlags)
838 {
839     bool rc = true
840
841     CAudioSignal *pSignals[2]
842     pSignals[0] = (CAudioSignal*)(pSignalsIn[0])
843     pSignals[1] = (CAudioSignal*)(pSignalsIn[1])
844
845     Free()
846
847     mProcessData = *pProcessData
848     mChannels = min(pSignals[0]->mNumChannels, pSignals[1]->mNumChannels)
849     mStepSize = mProcessData.mWindowSize - mProcessData.mOverlap
850
851     if (rc)
852     {
853         int s, c
854         OPTTRY
855         {
856             for (s=0 rc && s<mProcessData.mNumSignals s++)
857             {
858                 if (pSegments[s].Start<0) rc=false
859                 if (pSegments[s].End<0) rc=false
860                 if (pSegments[s].End>pSignals[s]->mSignalLength) rc=false
861                 if (pSegments[s].Start>pSignals[s]->mSignalLength-1) rc=false
862                 if (pSegments[s].End-pSegments[s].Start<mProcessData.mWindowSize) rc =false
863                 for (c=0 rc && c<mChannels c++)
864                 {
865                     rc = CalcFeatureForOneChannel(VecIndexFromChannel(s, c), pSignals[s], c,
&pSegments[s], s == 1 ? pActiveFrameFlags : 0)

```

```
866         }
867
868     }
869 }
870 OPTCATCH((...))
871 {
872     if (mProcessData.mpLogFile)
873         fprintf(mProcessData.mpLogFile, "\n*\n*\n***** ERROR ***** Caught unhandled
exception in CEnergyFeature::CalcVector(), signal=%d, channel=%d\n*\n*\n", s, c)
874         DebugBreak()
875     }
876 }
877 else Free()
878
879 return rc
880 }
881
882 CFeatureVector* CEnergyFeature::GetFVector(int IndexOfVector)
883 {
884     return (mpFeatureVectors[IndexOfVector])
885 }
886
887 }
888
```