```

    typedef double XFLOAT
    typedef double OTA_FLOAT

    typedef double OTA_FLOAT
    typedef MAT_DCplx OTA_CPLX


{

typedef struct
{
    float FrameWeightWeight
    bool  UseRelDistance
    float ViterbiDistanceWeightFactor
} VITERBI_PARA

typedef struct
{
    long Samplerate
    int mSRDetectFineAlignCorrlen
    int mDelayFineAlignCorrlen
    int WindowSize[8]
    int CoarseAlignCorrlen[8]
    float pViterbiDistanceWeightFactor[8]
} SPEECH_WINDOW_PARA

typedef struct
{
    SPEECH_WINDOW_PARA Win[3]
    float LowEnergyThresholdFactor
    float LowCorrelThreshold

    float FineAlignLowEnergyThresh
    float FineAlignLowEnergyCorrel
    float FineAlignShortDropOfCorrelR
    float FineAlignShortDropOfCorrelRLastBest
    float ViterbiDistanceWeightFactorDist
    float ViterbiDistanceWeightFactor
} SPEECH_TA_PARA

typedef struct
{
    SPEECH_WINDOW_PARA Win[3]
    float LowEnergyThresholdFactor
    float LowCorrelThreshold

    float FineAlignLowEnergyThresh
    float FineAlignLowEnergyCorrel
    float FineAlignShortDropOfCorrelR
    float FineAlignShortDropOfCorrelRLastBest
    float ViterbiDistanceWeightFactorDist
    float ViterbiDistanceWeightFactor
} AUDIO_TA_PARA

typedef struct
{
    float mCorrForSkippingInitialDelaySearch
    int CoarseAlignSegmentLengthInMs
} GENERAL_TA_PARA

typedef struct
{
    void Init(long Samplerate)
    {
        if (Samplerate==16000)      MaxWin=4
        else if (Samplerate==8000)  MaxWin=4
```

```
        else                        MaxWin=4

        LowPeakEliminationThreshold= 0.2000000029802322

        if (Samplerate==16000)      PercentageRequired = 0.05F
        else if (Samplerate==8000)  PercentageRequired = 0.1F
        else                        PercentageRequired = 0.02F

        MaxDistance = 14

        MinReliability = 7

        PercentageRequired = 0.7
        OTA_FLOAT MaxGradient = 1.1
        OTA_FLOAT MaxTimescaling = 0.1

        if (Samplerate==48000)      MaxStepPerFrame = MaxGradient * 1024.0
        else if (Samplerate==8000)  MaxStepPerFrame = MaxGradient * 128.0
        MaxBins = ((int)(MaxStepPerFrame*2.0*0.9))
        MaxStepPerFrame *= 4

    }

    float LowEnergyThresholdFactor
    float LowCorrelThreshold

    int     MaxStepPerFrame
    int     MaxBins
    int     MaxWin
    int     MinHistogramData

    float   MinReliability

    double  LowPeakEliminationThreshold
    float   MinFrequencyOfOccurrence
    float   LargeStepLimit

    float   MaxDistanceToLast
    float   MaxDistance
    float   MaxLargeStep

    float   ReliabilityThreshold
    float   PercentageRequired

    float   AllowedDistancePara2
    float   AllowedDistancePara3
} SR_ESTIMATION_PARA

class CParameters
{
    public:
        CParameters()
        {
            int i
            mTAPara.mCorrForSkippingInitialDelaySearch = 0.6F
            mTAPara.CoarseAlignSegmentLengthInMs = 600

            SPEECH_WINDOW_PARA      SpeechWinPara[] =
            {
                    {8000,   32, 32,
                        {128,  256, 128,  64,  32,  0, 0},
                        {-1,    -1,  -1,  86,  34,  0,  0},
                        {-1,    -1,  -1,  15,  12,  0,  0}},
                    {16000,  64, 64,
                        {256,  512, 256, 128,  64,  0},
                        {-1,    -1,  -1,  63,  33,  0},
                        {-1,    -1,  -1,  13,  10,  0}},
                    {48000, 256, 256,
```

```
                        {512, 1024, 512, 512, 128,  0},
                        {-1,    -1,  -1, 115,  61,  0},
                        {-1,    -1,  -1,  17,  16,  0}}
            }

            for (i=0  i<3  i++)
            {
                mSpeechTAPara.Win[i].Samplerate = SpeechWinPara[i].Samplerate
                mSpeechTAPara.Win[i].mDelayFineAlignCorrlen = SpeechWinPara[i].mDelayFineAlignCorrlen
                mSpeechTAPara.Win[i].mSRDetectFineAlignCorrlen =
SpeechWinPara[i].mSRDetectFineAlignCorrlen
                for (int k=0  k<8  k++)
                {
                    mSpeechTAPara.Win[i].CoarseAlignCorrlen[k] =
SpeechWinPara[i].CoarseAlignCorrlen[k]
                    mSpeechTAPara.Win[i].WindowSize[k]         = SpeechWinPara[i].WindowSize[k]

                    mSpeechTAPara.Win[i].pViterbiDistanceWeightFactor[k] =
SpeechWinPara[i].pViterbiDistanceWeightFactor[k]
                }
            }
            mSpeechTAPara.LowEnergyThresholdFactor = 15.0F
            mSpeechTAPara.LowCorrelThreshold = 0.4F
            mSpeechTAPara.FineAlignLowEnergyThresh = 2.0
            mSpeechTAPara.FineAlignLowEnergyCorrel = 0.6F
            mSpeechTAPara.FineAlignShortDropOfCorrelR = -1
            mSpeechTAPara.FineAlignShortDropOfCorrelRLastBest = 0.65F

            mSpeechTAPara.ViterbiDistanceWeightFactorDist = 5

            SPEECH_WINDOW_PARA      AudioWinPara[] =
            {
                    {8000,   32, 32,
                        {64,   128,  64,  64,  16,  0, 0},
                        {-1,    -1,  -1, 128,  32,  0, 0},
                        {-1,    -1,  -1,   6,   6,  0, 0}},
                    {16000,  64, 64,
                        {128,  256, 128, 128,  32,  0},
                        {-1,    -1,  -1,  64,  32,  0},
                        {-1,    -1,  -1,  12,  12,  0}},
                    {48000, 256, 2048,
                        {512, 1024, 512, 512, 256,  128,   0},
                        {-1,    -1,  -1, 512, 1024, 2048,  0},
                        {-1,    -1,  -1,  16,  16,  32,    0}}
            }

            for (i=0  i<3  i++)
            {
                mAudioTAPara.Win[i].Samplerate = AudioWinPara[i].Samplerate
                mAudioTAPara.Win[i].mDelayFineAlignCorrlen    =
AudioWinPara[i].mDelayFineAlignCorrlen
                mAudioTAPara.Win[i].mSRDetectFineAlignCorrlen =
AudioWinPara[i].mSRDetectFineAlignCorrlen
                for (int k=0  k<8  k++)
                {
                    mAudioTAPara.Win[i].CoarseAlignCorrlen[k] = AudioWinPara[i].CoarseAlignCorrlen[k]

                    mAudioTAPara.Win[i].WindowSize[k]         = AudioWinPara[i].WindowSize[k]
                    mAudioTAPara.Win[i].pViterbiDistanceWeightFactor[k] =
AudioWinPara[i].pViterbiDistanceWeightFactor[k]
                }
            }
            mAudioTAPara.LowEnergyThresholdFactor = 1
            mAudioTAPara.LowCorrelThreshold = 0.85F
            mAudioTAPara.FineAlignLowEnergyThresh = 32.0
            mAudioTAPara.FineAlignLowEnergyCorrel = 0.8F
            mAudioTAPara.FineAlignShortDropOfCorrelR = -1
            mAudioTAPara.FineAlignShortDropOfCorrelRLastBest = 0.8F
```

```
            mAudioTAPara.ViterbiDistanceWeightFactorDist = 6

            mSREPara.LowEnergyThresholdFactor = 15.0F
            mSREPara.LowCorrelThreshold = 0.4F

            mSREPara.MaxStepPerFrame = 160
            mSREPara.MaxBins = ((int)(mSREPara.MaxStepPerFrame*2.0*0.9))

            mSREPara.MaxWin=4
            mSREPara.LowPeakEliminationThreshold=0.2000000029802322F
            mSREPara.PercentageRequired = 0.04F

            mSREPara.LargeStepLimit = 0.08F
            mSREPara.MaxDistanceToLast = 7
            mSREPara.MaxLargeStep = 5
            mSREPara.MaxDistance = 14

            mSREPara.MinReliability = 7
            mSREPara.MinFrequencyOfOccurrence = 3

            mSREPara.AllowedDistancePara2 = 0.85F
            mSREPara.AllowedDistancePara3 = 1.5F

            mSREPara.ReliabilityThreshold = 0.3F
            mSREPara.MinHistogramData = 8

            mViterbi.UseRelDistance = false
            mViterbi.FrameWeightWeight = 1.0F
        }

        void Init(long Samplerate)
        {
            mSREPara.Init(Samplerate)
        }

        VITERBI_PARA        mViterbi
        GENERAL_TA_PARA     mTAPara
        SPEECH_TA_PARA      mSpeechTAPara
        AUDIO_TA_PARA       mAudioTAPara
        SR_ESTIMATION_PARA  mSREPara
}
}


{

class CProcessData
{
    public:
        CProcessData()
        {
            int i

            mCurrentIteration = -1
            mStartPlotIteration=10
            mLastPlotIteration =10
            mEnablePlotting=false
            mpLogFile = 0

            mWindowSize = 2048
            mSRDetectFineAlignCorrlen = 1024
            mDelayFineAlignCorrlen = 1024
            mOverlap    = 1024
            mSamplerate = 48000
            mNumSignals = 0
            mpMathlibHandle = 0
            mMinLowVarDelay = -99999999
            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
```

```
            D = mP.mSpeechTAPara.ViterbiDistanceWeightFactorDist
            D = D * mSamplerate / 1000
            F =((float) log(1+0.5) / (D*D))
            mP.mSpeechTAPara.ViterbiDistanceWeightFactor = F

            return true
        }

        CParameters   mP
}

class SECTION
{
    public:
        int Start
        int End
        int Len() {return End-Start }
        void CopyFrom(const SECTION &src)
        {
            this->Start = src.Start
            this->End   = src.End
        }
}

typedef struct OTA_RESULT
{
    void CopyFrom(const OTA_RESULT* src)
    {
        mNumFrames           = src->mNumFrames
        mStepsize            = src->mStepsize
        mResolutionInSamples = src->mResolutionInSamples
        if (src->mpDelay != NULL && mNumFrames > 0)
        {
            matFree(mpDelay)
            mpDelay = (long*)matMalloc(mNumFrames * sizeof(long))
            for (int i = 0  i < mNumFrames  i++)
                mpDelay[i] = src->mpDelay[i]
        }
        else
        {
            matFree(mpDelay)
            mpDelay = NULL
        }

        if (src->mpReliability != NULL && mNumFrames > 0)
        {
            matFree(mpReliability)
            mpReliability = (OTA_FLOAT*)matMalloc(mNumFrames * sizeof(OTA_FLOAT))
            for (int i = 0  i < mNumFrames  i++)
                mpReliability[i] = src->mpReliability[i]
        }
        else
        {
            matFree(mpReliability)
            mpReliability = NULL
        }
        mAvgReliability   = src->mAvgReliability
        mRelSamplerateDev = src->mRelSamplerateDev

        mNumUtterances = src->mNumUtterances
        if (src->mpStartSampleUtterance != NULL && mNumUtterances > 0)
        {
            matFree(mpStartSampleUtterance)
            mpStartSampleUtterance = (int*)matMalloc(mNumUtterances * sizeof(int))
            for (int i = 0  i < mNumUtterances  i++)
                mpStartSampleUtterance[i] = src->mpStartSampleUtterance[i]
        }
        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))
```

```
            for (int i = 0  i < mNumFrames  i++)
                mpActiveFrameFlags[i] = src->mpActiveFrameFlags[i]
        }
        else
        {
            matFree(mpActiveFrameFlags)
            mpActiveFrameFlags = NULL
        }

        if (src->mpIgnoreFlags != NULL && mNumFrames > 0)
        {

            matFree(mpIgnoreFlags)
            mpIgnoreFlags = (int*)matMalloc(mNumFrames * sizeof(int))
            mNumIngoreFlags = src->mNumIngoreFlags
            for (int i = 0  i < mNumFrames  i++)
                mpIgnoreFlags[i] = src->mpIgnoreFlags[i]
        }
        else
        {
            matFree(mpIgnoreFlags)
            mpIgnoreFlags = NULL
        }

        for (int i = 0  i < 5  i++)
            mTimeDiffs[i] = src->mTimeDiffs[i]

        mAslFrames = src->mAslFrames
        mAslFramelength = src->mAslFramelength
        if (src->mpAslActiveFrameFlags != NULL && mAslFrames > 0)
        {
            matFree(mpAslActiveFrameFlags)
            mpAslActiveFrameFlags = (int*)matMalloc(mAslFrames * sizeof(int))
            for (int i = 0  i < mAslFrames  i++)
                mpAslActiveFrameFlags[i] = src->mpAslActiveFrameFlags[i]
        }
        else
        {
            matFree(mpAslActiveFrameFlags)
            mpAslActiveFrameFlags = NULL
        }

        mAslFramesDeg = src->mAslFramesDeg
        if (src->mpAslActiveFrameFlagsDeg != NULL && mAslFramesDeg > 0)
        {
            matFree(mpAslActiveFrameFlagsDeg)
            mpAslActiveFrameFlagsDeg = (int*)matMalloc(mAslFramesDeg * sizeof(int))
            for (int i = 0  i < mAslFramesDeg  i++)
                mpAslActiveFrameFlagsDeg[i] = src->mpAslActiveFrameFlagsDeg[i]
        }
        else
        {
            matFree(mpAslActiveFrameFlagsDeg)
            mpAslActiveFrameFlagsDeg = NULL
        }

        FirstRefSample = src->FirstRefSample
        FirstDegSample = src->FirstDegSample
    }

    OTA_RESULT()
    {
        mNumFrames = 0
        mpDelay = NULL

        mpReliability = NULL

        mNumUtterances = 0
```

```
        mpStartSampleUtterance = NULL
        mpStopSampleUtterance  = NULL
        mpDelayUtterance       = NULL

        mNumSections = 0
        mpRefSections = NULL
        mpDegSections = NULL

        mpActiveFrameFlags = NULL
        mpIgnoreFlags = NULL
        mNumIngoreFlags = 0

        mAslFramelength = 0
        mAslFrames = 0
        mpAslActiveFrameFlags = NULL
        mAslFramesDeg = 0
        mpAslActiveFrameFlagsDeg = NULL

        FirstRefSample = FirstDegSample = 0
    }

    ~OTA_RESULT()
    {
        matFree(mpDelay)
        mpDelay = NULL

        matFree(mpReliability)
        mpReliability = NULL

        matFree(mpStartSampleUtterance)
        mpStartSampleUtterance = NULL

        matFree(mpStopSampleUtterance)
        mpStopSampleUtterance  = NULL

        matFree(mpDelayUtterance)
        mpDelayUtterance       = NULL

        delete[] mpRefSections
        mpRefSections = NULL
        delete[] mpDegSections
        mpDegSections = NULL

        matFree(mpActiveFrameFlags)
        mpActiveFrameFlags = NULL

        matFree(mpIgnoreFlags)
        mpIgnoreFlags = NULL

        matFree(mpAslActiveFrameFlags)
        mpAslActiveFrameFlags = NULL
        matFree(mpAslActiveFrameFlagsDeg)
        mpAslActiveFrameFlagsDeg = NULL
    }

    long mNumFrames
    int mStepsize
    int mResolutionInSamples
    int mPitchFrameSize
    long *mpDelay
    OTA_FLOAT *mpReliability
    OTA_FLOAT mAvgReliability
    OTA_FLOAT mRelSamplerateDev

    int  mNumUtterances
    int* mpStartSampleUtterance
    int* mpStopSampleUtterance
    int* mpDelayUtterance
```

```
    int FirstRefSample
    int FirstDegSample

    int         mNumSections
    SECTION     *mpRefSections
    SECTION     *mpDegSections

    double mSNRRefdB, mSNRDegdB
    double mNoiseLevelRef, mNoiseLevelDeg
    double mSignalLevelRef, mSignalLevelDeg
    double mNoiseThresholdRef, mNoiseThresholdDeg

    int *mpActiveFrameFlags

    int *mpIgnoreFlags
    int mNumIngoreFlags
    int mAslFrames
    int mAslFramelength
    int *mpAslActiveFrameFlags
    int mAslFramesDeg
    int *mpAslActiveFrameFlagsDeg

    double mTimeDiffs[5]

}OTA_RESULT

struct FilteringParameters
{
    int pListeningCondition
    double cutOffFrequencyLow
    double cutOffFrequencyHigh
    double disturbedEnergyQuotient
}

class ITempAlignment
{
    public:

        virtual bool Init(CProcessData* pProcessData)=0
        virtual void Free()=0
        virtual void Destroy()=0

        virtual bool SetSignal(int Index, unsigned long SampleRate, unsigned long NumSamples, int
NumChannels, OTA_FLOAT** pSignal)=0

        virtual void GetFilterCharacteristics(FilteringParameters *FilterParams)=0

        virtual bool FilterSignal(int Index, FilteringParameters *FilterParams)=0

        virtual bool Run(unsigned long Control, OTA_RESULT* pResult, int TArunIndex)=0

        virtual void GetNoiseSwitching(OTA_FLOAT* pBGNSwitchingLevel, OTA_FLOAT*
pNoiseLevelSpeechDeg, OTA_FLOAT* pNoiseLevelSilenceDeg)=0

        virtual OTA_FLOAT GetPitchFreq(int Signal, int Channel)=0

        virtual OTA_FLOAT GetPitchVector(int Signal, int Channel, OTA_FLOAT* pVector, int NumFrames,
int SamplesPerFrame)=0
        virtual int GetPitchFrameSize()=0
}

enum AlignmentType
{
    TA_FOR_SPEECH=0,

}

ITempAlignment* CreateAlignment(AlignmentType Type)
```

```
670
671 }
672
673
674 {
675
676 //Create the correlation matrix
677 //The structure is as follows:
678 //
679 // - This vector contains for each element of the underlying feature vectors
680 //   one vector with the correlation of all possible delay lags between mMinLowVarDelay and
    mMaxHighVarDelay.
681 //bool CDelaySearch::CreateMatrix(CFeatureList* FeatureList, CCAIntermediateResults* pCAIntermediate,
    CProcessData* pProcessData, int NumMacroFrames, int DegStep)
682 {
683     int f
684     bool rc = true
685
686     int* pActiveFrameFlags = pCAIntermediate->pActiveFrameFlags
687     long* pAvgDelayInFrames = pCAIntermediate->pDelayVec
688
689     if (mpCorrMatrix) Free()
690
691     mProcessData = *pProcessData
692     mpFeatureList = FeatureList
693     mNumFeatures  = mpFeatureList->mNumFeatures
694     mpChannels = new int[mNumFeatures]
695     if (!mpChannels) return false
696
697     mpCorrMatrix = new CCorrelationMatrix*[4*mNumFeatures]
698     if (mpCorrMatrix)
699     {
700         for (f=0  rc && f<4*mNumFeatures  f++)
701             mpCorrMatrix[f] = 0
702
703         for (f=0  rc && f<4*mNumFeatures  f++)
704         {
705             mpChannels[f] = mpFeatureList->GetChannels(f)
706             mpCorrMatrix[f] = new CCorrelationMatrix[mpChannels[f]]
707             rc = (mpCorrMatrix[f]!=0)
708         }
709     }
710     else rc = false
711
712     if (!rc) Free()
713
714     for (int k=0  k<2*mNumFeatures  k++)
715
716         if (rc)
717             for (int f=0  rc && f<mNumFeatures  f++)
718                 for (int c=0  rc && c<mpChannels[f]  c++)
719                     rc = mpCorrMatrix[f][c].CreateMatrix(mpFeatureList->mpFeatures[f], c,
    pActiveFrameFlags, &mProcessData, NumMacroFrames,
    pCAIntermediate->pSearchRangeLow, pCAIntermediate->pSearchRangeHigh,
    pCAIntermediate->pPitchVec, DegStep, pAvgDelayInFrames, f)
720
721     return rc
722 }
723
724 void CDelaySearch::MarkConstDelaySections(CCAIntermediateResults* pCAIntermediate, int DegStep)
725 {
726     int NumFrames = pCAIntermediate->mNumFrames
727     int* pActiveFrameFlags = pCAIntermediate->pActiveFrameFlags
728     long *pDelayVec  = pCAIntermediate->pDelayVec
729     int *pOptOffset  = pCAIntermediate->pOptOffset
730     int *pConstDelayMarker  = pCAIntermediate->pConstDelayMarker
731     MarkConstDelaySections(NumFrames, pActiveFrameFlags, pDelayVec, pOptOffset, pConstDelayMarker,
    DegStep)
```

```
}

void CDelaySearch::MarkConstDelaySections(CFAIntermediateResults* pFAIntermediate)
{
    int NumFrames = pFAIntermediate->mNumFrames
    int* pActiveFrameFlags = pFAIntermediate->pActiveFrameFlags
    long *pDelayVec  = pFAIntermediate->pDelayVec
    int *pOptOffset  = pFAIntermediate->pOptOffset
    int *pConstDelayMarker  = pFAIntermediate->pConstDelayMarker
    MarkConstDelaySections(NumFrames, pActiveFrameFlags, pDelayVec, pOptOffset, pConstDelayMarker, 1)
}

void CDelaySearch::MarkConstDelaySections(int NumFrames, int* pActiveFrameFlags, long *pDelayVec, int
*pOptOffset, int *pConstDelayMarker, int DegStep)
{

    int Marker = 10
    int LastConstDelay = pOptOffset[0]+pDelayVec[0]
    pConstDelayMarker[0] = pActiveFrameFlags[0] ? Marker : -1
    for (int i=1  i<NumFrames  i++)
    {
        pConstDelayMarker[i] = -1
        if (pActiveFrameFlags[i])
        {
            int Delay = pOptOffset[i]+pDelayVec[i]
            if (abs(Delay-LastConstDelay)>1)
            {
                LastConstDelay = Delay
                Marker++
                pConstDelayMarker[i] = Marker
            }
            pConstDelayMarker[i] = Marker
        }
    }

    Marker = 10
    int SectionStart=0
    int MinConst=MSecondsToFrames(100)/DegStep
    for (int i=1  i<NumFrames  i++)
    {
        if (pActiveFrameFlags[i]>0 && pActiveFrameFlags[i-1]>0)
        {
            if (pConstDelayMarker[i]!=Marker)
            {
                if (i-SectionStart<MinConst)
                {
                    if (pActiveFrameFlags[SectionStart])
                        for (int k=SectionStart  k<i  k++)
                            pConstDelayMarker[k] = -pConstDelayMarker[k]
                }
                Marker=pConstDelayMarker[i]
                SectionStart = i
            }
        }
    }
}

void UpdateConstDelay(int CurrentDelay, int* pLastDelays, int* pLastDelayPos)
{
    int Idx = *pLastDelayPos
    pLastDelays[Idx%1]= CurrentDelay
    *pLastDelayPos = Idx+1
}

int GetConstDelay(int CurrentDelay, int* pLastDelays, int* pLastDelayPos)
{
    int Idx = *pLastDelayPos
    int Delay = CurrentDelay
```

```
799
800     if (Idx>0)
801     {
802         const int MaxIdx = Idx%1+1
803         int Avg=0
804         for (int i=0  i<MaxIdx  i++)
805             Avg += pLastDelays[i]
806         Avg /= MaxIdx
807
808         if (Avg==CurrentDelay)
809             Delay = CurrentDelay
810         else
811         {
812             int BinDelay[1]
813             int BinFreq[1]
814             int LastBin=0
815             for (int i=0  i<MaxIdx  i++) {BinDelay[i]=0  BinFreq[i]=0 }
816
817             for (int i=0  i<MaxIdx  i++)
818             {
819                 Delay = pLastDelays[i]
820
821                 int b
822                 for (b=0  b<LastBin  b++)
823                     if (BinDelay[b]==Delay)
824                         break
825                 if (b==LastBin)
826                 {
827                     BinDelay[LastBin] = Delay
828                     BinFreq[LastBin++] = 1
829                 }
830                 else BinFreq[b]++
831             }
832
833             int MostLikelyIndex=0
834             int BestPeak = BinFreq[0]
835             for (int i=1  i<LastBin  i++)
836             {
837                 if (BinFreq[i]>BestPeak)
838                 {
839                     BestPeak = BinFreq[i]
840                     MostLikelyIndex = i
841                 }
842             }
843             int BestDelay = BinDelay[MostLikelyIndex]
844
845             int NumAvg=0
846             Delay = 0
847             for (int i=0  i<MaxIdx  i++)
848                 if (pLastDelays[i] < BestDelay+2 && pLastDelays[i] > BestDelay-2)
849                     {Delay += pLastDelays[i]  NumAvg++ }
850             Delay = ceil(Delay / (float)NumAvg)
851         }
852     }
853
854     return Delay
855 }
856
857 bool CDelaySearch::CalcOptimumPathThroughOneCorrelationMatrix2(int DegStep, CCAIntermediateResults*
    pCAIntermediate, CCorrelationMatrix* pMatrix,  OTA_FLOAT* pReliability)
858 {
859     int i
860     bool rc = true
861     long NumDegradedFrames = pMatrix->mNumMacroFrames
862     long NumRefFrames = pMatrix->mCorrelationVectorlength
863     long LastValidMaxPosInFF=0
864
865     int* pDelayOffsetPerFrame = pCAIntermediate->pRelativeDelayPerFrame
```

```
    int* FrameWithLastValidDelay = pCAIntermediate->pFrameWithLastValidDelay
    int* pActiveFrameFlags = pCAIntermediate->pActiveFrameFlags
    int *pOptOffset  = pCAIntermediate->pOptOffset

        int LastDelays[1]
    int LastDelayPos=0
    for (int j=0  j<1  j++)
        LastDelays[j] = NumRefFrames/2.0
    int OneMFInFF = DegStep

    OTA_FLOAT* PenaltyWeightFactor=pCAIntermediate->pPenaltyWeight

    int FirstActiveFrame=-1
    for (i=0  i<NumDegradedFrames  i++)
    {
        if (FirstActiveFrame<0 && pActiveFrameFlags[i])
        {
            FirstActiveFrame = i
        }

        if (FirstActiveFrame>=0)
        {
            OTA_FLOAT Distance = DegStep*FramesToMSeconds(i-FrameWithLastValidDelay[i])
            OTA_FLOAT WeightedDistance = 2.0e-6 * (Distance * Distance)

            PenaltyWeightFactor[i] = ((((OTA_FLOAT)1.0 - WeightedDistance) > ((OTA_FLOAT)0.0)) ?
((OTA_FLOAT)1.0 - WeightedDistance) : ((OTA_FLOAT)0.0))

            //Allow for very fast adaptations at the beginning of speech
            //Needs to be added here:
            //At the beginning of each active section and for the very first frame set the penalty
weight factor to 0.

            int FirstMaxIndex = pCAIntermediate->pMaxPositions[i]
            OTA_FLOAT FirstMaxR = pCAIntermediate->pMaxCorrelations[i]

            pMatrix->mpCorrMatrix[i][FirstMaxIndex] = -FirstMaxR
            int SecondMaxIndex
            OTA_FLOAT SecondMaxR = matMaxExt(pMatrix->mpCorrMatrix[i], NumRefFrames, &SecondMaxIndex)
            pMatrix->mpCorrMatrix[i][FirstMaxIndex] = FirstMaxR

            OTA_FLOAT Threshold = (((FirstMaxR*0.95) < (0.85)) ? (FirstMaxR*0.95) : (0.85))
            int NumAboveThreshold=0
            for (int k=0  k<NumRefFrames  k++)
                if (pMatrix->mpCorrMatrix[i][k]>Threshold)
                    NumAboveThreshold++
            bool IsBroadDistribution=false
            if (NumAboveThreshold>0.015*NumRefFrames)
                IsBroadDistribution = true

            int ConstDelay = ceil(FirstMaxIndex-NumRefFrames/2.0+pCAIntermediate->pDelayVec[i])
            if (pActiveFrameFlags[i])
                ConstDelay = GetConstDelay(ConstDelay, LastDelays, &LastDelayPos)
            else if (i>0)
                ConstDelay=LastValidMaxPosInFF-NumRefFrames/2.0-pCAIntermediate->pDelayVec[i]

            int IndexForConstDelay=(((NumRefFrames-1) < ((((0) >
(ConstDelay-pCAIntermediate->pDelayVec[i]+NumRefFrames/2.0)) ? (0) :
(ConstDelay-pCAIntermediate->pDelayVec[i]+NumRefFrames/2.0)))) ? (NumRefFrames-1) :
((((0) > (ConstDelay-pCAIntermediate->pDelayVec[i]+NumRefFrames/2.0)) ? (0) :
(ConstDelay-pCAIntermediate->pDelayVec[i]+NumRefFrames/2.0))))
            pCAIntermediate->pConstDelayIndex[i] = IndexForConstDelay

            //To be implemented here:
            //If the max correlation for this frame is larger than 0.97,
            //then set the penalty weight factor to 0.2.

            if (pActiveFrameFlags[i] && FrameWithLastValidDelay[i]>0)
```

```
928             {
929                 const int MaxDelayChange = OneMFInFF
930                 if ( (IndexForConstDelay-FirstMaxIndex) > MaxDelayChange)
931                 {
932                     if (IndexForConstDelay>=0 && IndexForConstDelay<NumRefFrames)
933                     {
934 
935                         PenaltyWeightFactor[i]*= 1.4
936 
937                         pCAIntermediate->pOptionsApplied[i]|=APPL_PATH_SEARCH_6
938                     }
939                 }
940             }
941 
942             if (pActiveFrameFlags[i] && abs(pDelayOffsetPerFrame[i])>=0.5*NumRefFrames/2)
943             {
944                 PenaltyWeightFactor[i] = 0.1
945                 pCAIntermediate->pOptionsApplied[i]|=APPL_CA_LOWER_PENALTY_FOR_JUMPS
946             }
947 
948             if (pActiveFrameFlags[i] && i<NumDegradedFrames-2 && i>0)
949                 if (!pActiveFrameFlags[i+1] || !pActiveFrameFlags[i+2])
950                 {
951                     PenaltyWeightFactor[i] = 1.0
952 
953                     pCAIntermediate->pOptionsApplied[i]|=APPL_PATH_SEARCH_4
954                 }
955 
956             LastValidMaxPosInFF = FirstMaxIndex
957             pCAIntermediate->pMaxPositions[i] = FirstMaxIndex
958             pCAIntermediate->pMaxCorrelations[i] = FirstMaxR
959 
960             if (pActiveFrameFlags[i])
961                 UpdateConstDelay(ceil(FirstMaxIndex-NumRefFrames/2.0+pCAIntermediate->pDelayVec[i]),
    LastDelays, &LastDelayPos)
962 
963             //If this is the first active frame, then we have to use all data for the frames since
    start as well.
964 
965             //This needs to be added to the public code
966 
967         }
968     }
969 
970     if (1 )
971     {
972         for (i=NumDegradedFrames-1  i>=0 && !pActiveFrameFlags[i]  i--)
973         {
974             matbCopy(pMatrix->mpCorrMatrix[FrameWithLastValidDelay[i]], pMatrix->mpCorrMatrix[i],
    NumRefFrames)
975             PenaltyWeightFactor[i] = 1.0
976         }
977     }
978 
979         //What is not published here:
980     //Set the weight factor for the first ten frames (which are not silence) to 0.
981 
982     rc = Viterbi(pMatrix->mpCorrMatrix, pDelayOffsetPerFrame, PenaltyWeightFactor, pOptOffset,
    pReliability, NumDegradedFrames, NumRefFrames, &mProcessData.mP.mViterbi)
983 
984     return rc
985 }
986 
987 bool CDelaySearch::CalcOptimumPathThroughOneCorrelationMatrix(int DegStep, CCAIntermediateResults*
    pCAIntermediate, CCorrelationMatrix* pMatrix, OTA_FLOAT* pReliability)
988 {
989 
990     return CalcOptimumPathThroughOneCorrelationMatrix2(DegStep, pCAIntermediate, pMatrix,
```

```
    pReliability)
991
992 }
993
994 }
995
```