```
    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

            mMinStaticDelayInMs = -2500
            mMaxStaticDelayInMs = 2500

            mMaxToleratedRelativeSamplerateDifference = 1.0

            for (i=0  i<8  i++)
                mpViterbiDistanceWeightFactor[i] = 0.0001F
        }

        int mMinStaticDelayInMs
        int mMaxStaticDelayInMs

        int mMinLowVarDelayInSamples
        int mMaxHighVarDelayInSamples

        int mStartPlotIteration
        int mLastPlotIteration
        bool mEnablePlotting
        long mSamplerate

        FILE* mpLogFile

        int mCurrentIteration

        int mpWindowSize[8]

        int mpOverlap[8]

        int mpCoarseAlignCorrlen[8]

        float mpViterbiDistanceWeightFactor[8]

        int mDelayFineAlignCorrlen
        int mSRDetectFineAlignCorrlen
        float mMaxToleratedRelativeSamplerateDifference
        int mWindowSize

        int mOverlap

        int mCoarseAlignCorrlen

        int mNumSignals
        void* mpMathlibHandle

        int mMinLowVarDelay
        int mMaxHighVarDelay
        int mStepSize

        bool Init(int Iteration, float MoreDownsampling)
        {
            assert(MoreDownsampling)

            mCurrentIteration = Iteration
            mP.Init(mSamplerate)

            mWindowSize = (int)((float)mpWindowSize[Iteration]*MoreDownsampling)
            mOverlap = (int)((float)mpOverlap[Iteration]*MoreDownsampling)
            mCoarseAlignCorrlen = mpCoarseAlignCorrlen[Iteration]
            mStepSize = mWindowSize - mOverlap
            mMinLowVarDelay = mMinLowVarDelayInSamples / mStepSize
            mMaxHighVarDelay = mMaxHighVarDelayInSamples / mStepSize

            float D = mpViterbiDistanceWeightFactor[Iteration]
            D = D * mSamplerate / mStepSize / 1000
            float F = ((float)log(1+0.5)) / (D*D)
            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         }
```

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

        mNumSections = src->mNumSections
        if (src->mpRefSections != NULL && mNumSections > 0)
        {
            delete[] mpRefSections
            mpRefSections = new SECTION[mNumSections]
            for (int i = 0  i < mNumSections  i++)
                mpRefSections[i].CopyFrom(src->mpRefSections[i])
        }
        else
        {
            delete[] mpRefSections
            mpRefSections = NULL
        }
        if (src->mpDegSections != NULL && mNumSections > 0)
        {
            delete[] mpDegSections
            mpDegSections = new SECTION[mNumSections]
            for (int i = 0  i < mNumSections  i++)
                mpDegSections[i].CopyFrom(src->mpDegSections[i])
        }
        else
        {
            delete[] mpDegSections
            mpDegSections = NULL
        }

        mSNRRefdB = src->mSNRRefdB
        mSNRDegdB = src->mSNRDegdB
        mNoiseLevelRef = src->mNoiseLevelRef
        mNoiseLevelDeg = src->mNoiseLevelDeg
        mSignalLevelRef = src->mSignalLevelRef
        mSignalLevelDeg = src->mSignalLevelDeg
        mNoiseThresholdRef = src->mNoiseThresholdRef
        mNoiseThresholdDeg = src->mNoiseThresholdDeg

        if (src->mpActiveFrameFlags != NULL && mNumFrames > 0)
        {
            matFree(mpActiveFrameFlags)
```

```
            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,

}
```

```
669 ITempAlignment* CreateAlignment(AlignmentType Type)
670
671 }
672
673
674 {
675
676 void GetNormalizedCCF(MAT_HANDLE mh, const OTA_FLOAT*  srcA,  int lenA, const OTA_FLOAT*  srcB, int
    lenB, OTA_FLOAT*  dst, int dstLen)
677 void GetNormalizedCCFHistogram(MAT_HANDLE mh, const OTA_FLOAT*  srcA,  int lenA, const OTA_FLOAT*
    srcB, int lenB, OTA_FLOAT*  dst, int dstLen, int HistoLen)
678 void GetNormalizedCCFPeakHistogram(MAT_HANDLE mh, const OTA_FLOAT*  srcA,  int lenA, const
    OTA_FLOAT*  srcB, int lenB, OTA_FLOAT*  dst, int dstLen, int HistoLen)
679
680 extern FILE* pLogFile
681
682 inline void NaNTest2(const OTA_FLOAT* Vec, const int Len)
683 {
684     int i
685     for (i=0  i<Len  i++)
686         assert(Vec[i]==Vec[i])
687 }
688
689 void
690 {
691 }
692
693 void SmoothHistogramTriangular(MAT_HANDLE mh, OTA_FLOAT* pHistogram, int HistogramLen, int
    KernelWidth)
694 {
695     int i
696     int Center = KernelWidth / 2
697     KernelWidth = 2*Center+1
698     OTA_FLOAT* pKernel = matxMalloc(KernelWidth)
699     for (i=1  i<Center  i++)
700     {
701         OTA_FLOAT NextVal = (OTA_FLOAT)i/(OTA_FLOAT)Center
702         pKernel[i] = NextVal
703         pKernel[KernelWidth-i-1] = NextVal
704     }
705     pKernel[0] = pKernel[KernelWidth-1] = 0
706     pKernel[Center] = 1
707
708     matRunFIRFilter(mh, pHistogram, pHistogram, HistogramLen, pKernel, KernelWidth,
    MAT_FIRDelayComp)
709
710     matFree(pKernel)
711
712 }
713
714 inline void GetNormalizedCCFCore(MAT_HANDLE mh, int FFTlen, int order, const OTA_FLOAT* srcA, const
    OTA_FLOAT* srcB, OTA_FLOAT* dst, int lenA, int lenB, int dstLen, OTA_FLOAT* tempinA, OTA_FLOAT*
    tempinB, OTA_FLOAT* tempinC, OTA_CPLX* tempoutA, OTA_CPLX* tempoutB, OTA_CPLX* tempoutC)
715 {
716     for (int d=0  d<dstLen  d++)
717         dst[d] += matPearsonCorrelation((OTA_FLOAT*)srcA+d, (OTA_FLOAT*)srcB, lenB)
718
719     NaNTest(dst, dstLen)
720 }
721
722 void GetNormalizedCCF(MAT_HANDLE mh, const OTA_FLOAT*  srcA,  int lenA, const OTA_FLOAT*  srcB, int
    lenB, OTA_FLOAT*  dst, int dstLen)
723 {
724     int order=0
725     OTA_FLOAT* tempinA=NULL
726     OTA_FLOAT* tempinB=NULL
727     OTA_FLOAT* tempinC=NULL
728     OTA_FLOAT* tempintest=NULL
```

```
    OTA_CPLX* tempoutA=NULL
    OTA_CPLX* tempoutB=NULL
    OTA_CPLX* tempoutC=NULL

    int MinLen = dstLen+lenB
    while (1<<order <= MinLen)
        order++
    order++
    int FFTlen = 1<<order
    lenA = (((lenA) < (FFTlen/2)) ? (lenA) : (FFTlen/2))

    tempinA=(OTA_FLOAT*)matMalloc (sizeof (OTA_FLOAT)*FFTlen)
    tempinB=(OTA_FLOAT*)matMalloc (sizeof (OTA_FLOAT)*FFTlen)
    tempinC=(OTA_FLOAT*)matMalloc (sizeof (OTA_FLOAT)*(FFTlen+2))

    tempoutA= (OTA_CPLX*)matMalloc (sizeof(OTA_CPLX)*(FFTlen/2+1))
    tempoutB= (OTA_CPLX*)matMalloc (sizeof(OTA_CPLX)*(FFTlen/2+1))
    tempoutC= (OTA_CPLX*)matMalloc (sizeof(OTA_CPLX)*(FFTlen/2+1))

    matbSet(0.0, dst, dstLen)
    GetNormalizedCCFCore(mh, FFTlen, order, srcA, srcB, dst, lenA, lenB, dstLen, tempinA, tempinB,
tempinC, tempoutA, tempoutB, tempoutC)

    matFree(tempoutC)
    matFree(tempoutB)
    matFree(tempoutA)
    matFree(tempinB)
    matFree(tempinA)
    matFree(tempinC)

}

void GetNormalizedCCFHistogram(MAT_HANDLE mh, const OTA_FLOAT*  srcA,  int lenA, const OTA_FLOAT*
srcB, int lenB, OTA_FLOAT*  dst, int dstLen, int HistoLen)
{
    int order=0
    OTA_FLOAT* tempinA=NULL
    OTA_FLOAT* tempinB=NULL
    OTA_FLOAT* tempinC=NULL
    OTA_FLOAT* tempintest=NULL
    OTA_CPLX* tempoutA=NULL
    OTA_CPLX* tempoutB=NULL
    OTA_CPLX* tempoutC=NULL

    int i

    int MinLen = dstLen+lenB
    while (1<<order <= MinLen)
        order++
    order++
    int FFTlen = 1<<order
    lenA = (((lenA) < (FFTlen/2)) ? (lenA) : (FFTlen/2))

    tempinA=(OTA_FLOAT*)matMalloc (sizeof (OTA_FLOAT)*FFTlen)
    tempinB=(OTA_FLOAT*)matMalloc (sizeof (OTA_FLOAT)*FFTlen)
    tempinC=(OTA_FLOAT*)matMalloc (sizeof (OTA_FLOAT)*FFTlen+2)

    tempoutA= (OTA_CPLX*)matMalloc (sizeof(OTA_CPLX)*(FFTlen/2+1))
    tempoutB= (OTA_CPLX*)matMalloc (sizeof(OTA_CPLX)*(FFTlen/2+1))
    tempoutC= (OTA_CPLX*)matMalloc (sizeof(OTA_CPLX)*(FFTlen/2+1))

    matbSet(0.0, dst, dstLen)
    for (i=0  i<HistoLen  i++)
        GetNormalizedCCFCore(mh, FFTlen, order, srcA+i, srcB+i, dst, lenA, lenB, dstLen, tempinA,
tempinB, tempinC, tempoutA, tempoutB, tempoutC)

    for (i=0  i<dstLen  i++)
        dst[i] /= HistoLen
```

```
794
795     matFree(tempoutC)
796     matFree(tempoutB)
797     matFree(tempoutA)
798     matFree(tempinB)
799     matFree(tempinA)
800     matFree(tempinC)
801
802 }
803
804 void GetNormalizedCCFPeakHistogram(MAT_HANDLE mh, const OTA_FLOAT*  srcA,  int lenA, const
    OTA_FLOAT*  srcB, int lenB, OTA_FLOAT*  dst, int dstLen, int HistoLen)
805 {
806     int order=0
807     OTA_FLOAT* Correl=NULL
808     OTA_FLOAT* tempinA=NULL
809     OTA_FLOAT* tempinB=NULL
810     OTA_FLOAT* tempinC=NULL
811     OTA_FLOAT* tempintest=NULL
812     OTA_CPLX* tempoutA=NULL
813     OTA_CPLX* tempoutB=NULL
814     OTA_CPLX* tempoutC=NULL
815
816     int i
817
818     int MinLen = dstLen+lenB
819     while (1<<order <= MinLen)
820         order++
821     order++
822     int FFTlen = 1<<order
823     lenA = (((lenA) < (FFTlen/2)) ? (lenA) : (FFTlen/2))
824
825     Correl=matxMalloc(dstLen)
826     tempinA=(OTA_FLOAT*)matMalloc (sizeof (OTA_FLOAT)*FFTlen)
827     tempinB=(OTA_FLOAT*)matMalloc (sizeof (OTA_FLOAT)*FFTlen)
828     tempinC=(OTA_FLOAT*)matMalloc (sizeof (OTA_FLOAT)*FFTlen+2)
829
830     tempoutA= (OTA_CPLX*)matMalloc (sizeof(OTA_CPLX)*(FFTlen/2+1))
831     tempoutB= (OTA_CPLX*)matMalloc (sizeof(OTA_CPLX)*(FFTlen/2+1))
832     tempoutC= (OTA_CPLX*)matMalloc (sizeof(OTA_CPLX)*(FFTlen/2+1))
833
834     int MaxIndex=-1
835     for (i=0  i<HistoLen  i++)
836     {
837         matbSet(0.0, dst, dstLen)
838         GetNormalizedCCFCore(mh, FFTlen, order, srcA+i, srcB+i, Correl, lenA, lenB, dstLen, tempinA,
    tempinB, tempinC, tempoutA, tempoutB, tempoutC)
839         OTA_FLOAT RMax=matMaxExt(Correl, dstLen, &MaxIndex)
840         dst[MaxIndex] += RMax
841     }
842     for (i=0  i<dstLen  i++)
843         dst[i] /= HistoLen
844
845     matFree(tempoutC)
846     matFree(tempoutB)
847     matFree(tempoutA)
848     matFree(tempinB)
849     matFree(tempinA)
850     matFree(tempinC)
851     matFree(Correl)
852
853 }
854
855 int FindDelay(MAT_HANDLE mh, OTA_FLOAT* pA, int LenA, OTA_FLOAT* pB, int LenB, int HistoLen, int
    HistoShift, int MaxDelay, OTA_FLOAT* pPearsonCorrelation, bool PlotMe)
856 {
857
858     int FoundDelay = -1
```

```
859
860     if(LenA > 0 && LenB > 0)
861     {
862         OTA_FLOAT Correl=-1
863         if (MaxDelay==0) MaxDelay = (((0) > (LenB-LenA-(HistoLen*HistoShift))) ? (0) :
    (LenB-LenA-(HistoLen*HistoShift)))
864         int KernelWidth = (((8) < (MaxDelay/2)) ? (8) : (MaxDelay/2))
865         OTA_FLOAT* pDest = (OTA_FLOAT*)matMalloc (sizeof (OTA_FLOAT)*MaxDelay)
866         OTA_FLOAT* pHistogramRel = (OTA_FLOAT*)matMalloc (sizeof (OTA_FLOAT)*MaxDelay)
867
868         int Offset = 0
869         matbSet(0.0, pHistogramRel, MaxDelay)
870
871         for (int i=0  i<HistoLen  i++)
872         {
873             int MaxDelayUsed = MaxDelay
874             if (MaxDelayUsed>0)
875             {
876
877                 for (int p=0  p<MaxDelay  p++)
878
879                     pDest[p] = matPearsonCorrelation(pA+Offset, pB+p+Offset, LenA)
880
881                 Correl = matMaxExt(pDest, MaxDelayUsed, &FoundDelay)
882
883                 if(FoundDelay >= 0)
884                     pHistogramRel[FoundDelay]+= Correl
885             }
886
887             Offset += HistoShift
888         }
889         if (HistoLen>1)
890             SmoothHistogramTriangular(mh, pHistogramRel, MaxDelay, KernelWidth)
891
892         matMaxExt(pHistogramRel, MaxDelay, &FoundDelay)
893
894         if (pPearsonCorrelation)
895         {
896             int Len = (((LenA) < (LenB-FoundDelay)) ? (LenA) : (LenB-FoundDelay))
897             if (Len>0)
898                 *pPearsonCorrelation = Correl = matPearsonCorrelation(pA, pB+FoundDelay, Len)
899             else
900                 *pPearsonCorrelation = 0
901
902         }
903
904         matFree(pDest)
905         matFree(pHistogramRel)
906
907     }
908     else
909     {
910         FoundDelay = 0
911         *pPearsonCorrelation = 0
912     }
913     return FoundDelay
914 }
915
916 int FindDelayStrict(MAT_HANDLE mh, OTA_FLOAT* pA, int LenA, OTA_FLOAT* pB, int LenB, int HistoLen,
    int HistoShift, int MaxDelay, OTA_FLOAT* pPearsonCorrelation)
917 {
918     int FoundDelay = -1
919
920     if(LenA > 0 && LenB > 0)
921     {
922         int SilenceOffset = 0
923
924         LenB -= SilenceOffset
```

```
925         pB += SilenceOffset
926
927         OTA_FLOAT Correl=-1
928         if (MaxDelay==0) MaxDelay = (((0) > (LenB-LenA)) ? (0) : (LenB-LenA))
929         int KernelWidth = (((8) < (MaxDelay/2)) ? (8) : (MaxDelay/2))
930         OTA_FLOAT* pDest = (OTA_FLOAT*)matMalloc (sizeof (OTA_FLOAT)*MaxDelay)
931         OTA_FLOAT* pHistogramRel = (OTA_FLOAT*)matMalloc (sizeof (OTA_FLOAT)*MaxDelay)
932
933         int Offset = 0
934         matbSet(0.0, pHistogramRel, MaxDelay)
935
936         for (int i=0  i<HistoLen  i++)
937         {
938             int MaxDelayUsed = (((0) > ((((MaxDelay) < (LenB-LenA-Offset)) ? (MaxDelay) :
    (LenB-LenA-Offset)))) ? (0) : ((((MaxDelay) < (LenB-LenA-Offset)) ? (MaxDelay) :
    (LenB-LenA-Offset))))
939             if (MaxDelayUsed>0)
940             {
941                 matCrossCorr(mh, pA+Offset, LenA, pB+Offset, LenB, pDest, MaxDelayUsed, 0)
942                 Correl = matMaxExt(pDest, MaxDelayUsed, &FoundDelay)
943
944                 if(FoundDelay >= 0)
945                     pHistogramRel[FoundDelay]+= Correl
946             }
947
948             Offset += HistoShift
949         }
950
951         if (HistoLen>1)
952             SmoothHistogramTriangular(mh, pHistogramRel, MaxDelay, KernelWidth)
953
954         matMaxExt(pHistogramRel, MaxDelay, &FoundDelay)
955
956         if (pPearsonCorrelation)
957         {
958             int Len = (((LenA) < (LenB-FoundDelay)) ? (LenA) : (LenB-FoundDelay))
959             if (Len>0)
960                 *pPearsonCorrelation = Correl = matPearsonCorrelation(pA, pB+FoundDelay, Len)
961             else
962                 *pPearsonCorrelation = 0
963         }
964
965         FoundDelay += SilenceOffset
966
967         matFree(pDest)
968         matFree(pHistogramRel)
969
970     }
971     else
972     {
973         FoundDelay = 0
974         *pPearsonCorrelation = 0
975     }
976     return FoundDelay
977 }
978
979 void FindMaxCorrelation(MAT_HANDLE mh, OTA_FLOAT* data1, unsigned long length1, OTA_FLOAT* data2,
    unsigned long length2, int low_lag, int high_lag, int* index, OTA_FLOAT* maximum, OTA_FLOAT*
    CorrelationBuffer)
980 {
981     unsigned long CorrLen = high_lag-low_lag
982
983     matbSet(0, CorrelationBuffer, CorrLen)
984     GetNormalizedCCF(mh, data1, length1, data2, length2, CorrelationBuffer, CorrLen)
985     *maximum=matMaxExt(CorrelationBuffer, CorrLen, index)
986
987     *maximum = (((1.0) < (*maximum)) ? (1.0) : (*maximum))
988     *maximum = (((-1.0) > (*maximum)) ? (-1.0) : (*maximum))
```

```
989
990 }
991
992 inline OTA_FLOAT A(int RFrom, int RTo, int MaxDistance, OTA_FLOAT* FromCorrelation, OTA_FLOAT*
    ToCorrelation, OTA_FLOAT Factor1, OTA_FLOAT Factor2)
993 {
994     OTA_FLOAT Distance = abs((RTo)-(RFrom))
995     if (Distance*Distance*Factor1>13.8)
996         return -1.0e6
997     else
998         return 1-exp(Distance*Distance*Factor1)
999 }
1000
1001 void ComputePenaltyTable(long Size, long Center, OTA_FLOAT WeightFactor, OTA_FLOAT* pPenaltyTable)
1002 {
1003     int d
1004     for (d=0  d<Size  d++)
1005     {
1006         pPenaltyTable[d] = d-Center
1007         if (d>Center)   pPenaltyTable[d]*=1
1008         else            pPenaltyTable[d]*=1
1009     }
1010     matbAbs1(pPenaltyTable, Size)
1011     matbSqr1(pPenaltyTable, Size)
1012     matbMpy1(WeightFactor, pPenaltyTable, Size)
1013     matbThresh1(pPenaltyTable, Size, 9.21034037197618273607196581873 75, MAT_GT)
1014     matbExp1(pPenaltyTable, Size)
1015     for (d=0  d<Size  d++)
1016     {
1017         pPenaltyTable[d] = 1e10*(1.0-pPenaltyTable[d])
1018         pPenaltyTable[d] = floor(pPenaltyTable[d])/1e10
1019     }
1020
1021 }
1022
1023
1024 bool Viterbi(OTA_FLOAT** pMatrix, int* pOffsetPerFrame, OTA_FLOAT* PenaltyWeightFactor, int
    *pOptOffset, OTA_FLOAT* pReliability, long NumDegradedFrames, long NumRefFrames, VITERBI_PARA*
    pPara)
1025 {
1026     bool rc = true
1027
1028     OTA_FLOAT** P
1029     int**   L
1030
1031
1032
1033     L = (int**)matMalloc2D(NumDegradedFrames, NumRefFrames * sizeof(int))
1034     P = (OTA_FLOAT**)matMalloc2D(NumDegradedFrames, NumRefFrames * sizeof(OTA_FLOAT))
1035
1036     if (L && P)
1037     {
1038         for (int i=0  rc && i<NumDegradedFrames  i++)
1039         {
1040             if (!P[i]) rc = false
1041             if (!L[i]) rc = false
1042         }
1043     }
1044     else rc = false
1045
1046     OTA_FLOAT* pPenaltyTable = (OTA_FLOAT*)matMalloc((2*NumRefFrames+1) * sizeof(OTA_FLOAT))
1047     ComputePenaltyTable(2*NumRefFrames+1, NumRefFrames, pPara->ViterbiDistanceWeightFactor,
    pPenaltyTable)
1048
1049     OTA_FLOAT *PathProb = (OTA_FLOAT*)matMalloc(NumRefFrames * sizeof(OTA_FLOAT))
1050
1051     if (rc && PathProb && pPenaltyTable)
1052     {
```

```
1053        int d
1054
1055        for (d=0  d<NumDegradedFrames  d++)
1056        {
1057            int i
1058
1059            matbCopy(pMatrix[d], P[d], NumRefFrames)
1060            matbThresh1(P[d], NumRefFrames, 0.0, MAT_LT)
1061            matbThresh1(P[d], NumRefFrames, 0.999, MAT_GT)
1062            for (i=0  i<NumRefFrames  i++)
1063                P[d][i] = (1-P[d][i])
1064            for (i=0  i<NumRefFrames  i++)
1065                P[d][i] = -log10(P[d][i])
1066
1067            OTA_FLOAT PMin = matMin(P[d], NumRefFrames)
1068            OTA_FLOAT PMax = matMax(P[d], NumRefFrames)
1069            if (PMin==PMax)
1070                P[d][NumRefFrames/2] = PMax + 0.1
1071
1072        }
1073
1074        for (d=1  d<NumDegradedFrames  d++)
1075        {
1076
1077            int LastMaxPos
1078            OTA_FLOAT LastMax
1079
1080            if (pPara->UseRelDistance)
1081            {
1082                for (int r=0  r<NumRefFrames  r++)
1083                {
1084
1085                    matbCopy(P[d-1], PathProb, NumRefFrames)
1086                    for (int rr=0  rr<NumRefFrames  rr++)
1087                    {
1088                        int TableIndex = ((((((r + pOffsetPerFrame[d]-rr+NumRefFrames) <
    (2*NumRefFrames)) ? (r + pOffsetPerFrame[d]-rr+NumRefFrames) :
    (2*NumRefFrames))) > (0)) ? ((((r + pOffsetPerFrame[d]-rr+NumRefFrames) <
    (2*NumRefFrames)) ? (r + pOffsetPerFrame[d]-rr+NumRefFrames) :
    (2*NumRefFrames))) : (0))
1089                        PathProb[rr] = PathProb[rr] + pPenaltyTable[TableIndex] *
    PenaltyWeightFactor[d]
1090                    }
1091
1092                    OTA_FLOAT LastMax = matMaxExt(PathProb, NumRefFrames, &LastMaxPos)
1093
1094                    L[d][r] = LastMaxPos
1095
1096                    P[d][r] = P[d][r] + LastMax
1097                }
1098            }
1099            else
1100            {
1101                for (int r=0  r<NumRefFrames  r++)
1102                {
1103
1104                    matbCopy(P[d-1], PathProb, NumRefFrames)
1105                    for (int rr=0  rr<NumRefFrames  rr++)
1106                    {
1107                        int TableIndex = ((((((r -rr+NumRefFrames) < (2*NumRefFrames)) ? (r
    -rr+NumRefFrames) : (2*NumRefFrames))) > (0)) ? ((((r -rr+NumRefFrames) <
    (2*NumRefFrames)) ? (r -rr+NumRefFrames) : (2*NumRefFrames))) : (0))
1108                        PathProb[rr] = PathProb[rr] + pPenaltyTable[TableIndex] *
    PenaltyWeightFactor[d]
1109                    }
1110
1111                    LastMax = matMaxExt(PathProb, NumRefFrames, &LastMaxPos)
1112
```

```
                    L[d][r] = LastMaxPos

                    P[d][r] = P[d][r] + LastMax
                }

            }
        }

        matMaxExt(P[NumDegradedFrames-1], NumRefFrames, pOptOffset+NumDegradedFrames-1)

        for (d=NumDegradedFrames-1  d>0  d--)
        {
            int LastOpt = pOptOffset[d]
            pOptOffset[d-1] = L[d][LastOpt]
        }

        for (d=0  d<NumDegradedFrames  d++)
            pReliability[d] = pMatrix[d][pOptOffset[d]]
    }

    if (PathProb)
        matFree(PathProb)
    if (pPenaltyTable)
        matFree(pPenaltyTable)

    matFree2D((void**)P)
    matFree2D((void**)L)

    return rc
}

OTA_FLOAT* matxMalloc(int len)
{
    if (sizeof(OTA_FLOAT)==sizeof(float))
        return (OTA_FLOAT*)matsMalloc(len)
    else
        return (OTA_FLOAT*)matdMalloc(len)
}

OTA_CPLX* matCplxMalloc(int len)
{
    if (sizeof(OTA_CPLX)==sizeof(MAT_SCplx))
        return (OTA_CPLX*)matcMalloc(len)
    else
        return (OTA_CPLX*)matzMalloc(len)
}

}
```