```


    typedef double XFLOAT
    typedef double OTA_FLOAT


{
    class CDelayPara
    {
        public:
            CDelayPara()
            {
                Init()
            }

            ~CDelayPara()
            {
                Free()
            }

            void operator= (const CDelayPara &Src)
            {
                Free()

                Framesize = Src.Framesize
                FramesUsed = Src.FramesUsed
                MaxModelFramesAlloc = 2*Src.MaxModelFramesAlloc
                MaxModelFrames = 2*Src.MaxModelFrames
                FramesUsed = Src.FramesUsed
                MaxSigLen = Src.MaxSigLen
                LogFile = Src.LogFile
                OriginalNumberOfSamples = Src.OriginalNumberOfSamples
                pOriginalSamples = Src.pOriginalSamples
                DistortedNumberOfSamples = Src.DistortedNumberOfSamples
                pDistortedSamples = Src.pDistortedSamples
                pStartSampleUtterance = Src.pStartSampleUtterance
                pStopSampleUtterance = Src.pStopSampleUtterance
                pDelayUtterance = Src.pDelayUtterance
                FirstRefSample = Src.FirstRefSample
                FirstDegSample = Src.FirstDegSample
                PitchFrameSize = Src.PitchFrameSize
                PitchFreqRef = Src.PitchFreqRef
                PitchFreqDeg = Src.PitchFreqDeg
                AslFramelength = Src.AslFramelength
                AslFrames = Src.AslFrames
                AslFramesDeg = Src.AslFramesDeg
                mh = Src.mh

                AllocVectors(Src.MaxModelFramesAlloc)
                pNoiseDuringSpeechdB[0] = Src.pNoiseDuringSpeechdB[0]
                pNoiseDuringSpeechdB[1] = Src.pNoiseDuringSpeechdB[1]
                pNoiseDuringSilencedB[0] = Src.pNoiseDuringSilencedB[0]
                pNoiseDuringSilencedB[1] = Src.pNoiseDuringSilencedB[1]
                pBGNSwitchingLevel[0] = Src.pBGNSwitchingLevel[0]
                pBGNSwitchingLevel[1] = Src.pBGNSwitchingLevel[1]

                matbCopy(Src.pPitchVecOfRef, pPitchVecOfRef, MaxModelFramesAlloc)
                matbCopy(Src.pPitchVecOfDeg, pPitchVecOfDeg, MaxModelFramesAlloc)
                matbCopy(Src.pDelayReliability, pDelayReliability, MaxModelFramesAlloc)
                matbCopy(Src.pIgnoreFrameFlags, pIgnoreFrameFlags, MaxModelFramesAlloc)

                memcpy(pActiveFrameFlags, Src.pActiveFrameFlags, sizeof(bool)*MaxModelFramesAlloc)
                memcpy(pAslActiveFrameFlags, Src.pAslActiveFrameFlags,
sizeof(bool)*MaxModelFramesAlloc)
                memcpy(pAslActiveFrameFlagsDeg, Src.pAslActiveFrameFlagsDeg,
sizeof(bool)*MaxModelFramesAlloc)
            }

```

```
        void Check()
        {
            if (PitchFreqRef != PitchFreqRef)
                DebugBreak()
            if (PitchFreqDeg != PitchFreqDeg)
                DebugBreak()
            if (pNoiseDuringSpeechdB[0] != pNoiseDuringSpeechdB[0])
                DebugBreak()
            if (pNoiseDuringSpeechdB[1] != pNoiseDuringSpeechdB[1])
                DebugBreak()
            if (pNoiseDuringSilencedB[0] != pNoiseDuringSilencedB[0])
                DebugBreak()
            if (pNoiseDuringSilencedB[1] != pNoiseDuringSilencedB[1])
                DebugBreak()
            if (pBGNSwitchingLevel[0] != pBGNSwitchingLevel[0])
                DebugBreak()
            if (pBGNSwitchingLevel[1] != pBGNSwitchingLevel[1])
                DebugBreak()

            for (int i=0  i<MaxModelFrames  i++)
                if (pPitchVecOfRef[i] != pPitchVecOfRef[i])
                    DebugBreak()
            for (int i=0  i<MaxModelFrames  i++)
                if (pPitchVecOfDeg[i] != pPitchVecOfDeg[i])
                    DebugBreak()
            for (int i=0  i<MaxModelFrames  i++)
                if (pDelayReliability[i] != pDelayReliability[i])
                    DebugBreak()

            if (FramesUsed<0 || FramesUsed>MaxModelFrames)
                DebugBreak()
            if (AslFrames<0 || AslFrames>MaxModelFrames)
                DebugBreak()
            if (AslFramesDeg<0 || AslFramesDeg>MaxModelFrames)
                DebugBreak()

            if (OriginalNumberOfSamples<0 || OriginalNumberOfSamples>MaxSigLen)
                DebugBreak()
            if (DistortedNumberOfSamples<0 || DistortedNumberOfSamples>MaxSigLen)
                DebugBreak()

            if (FirstRefSample<0 || FirstRefSample>OriginalNumberOfSamples)
                DebugBreak()
            if (FirstDegSample<0 || FirstDegSample>DistortedNumberOfSamples)
                DebugBreak()

            if (AslFramelength<0 || AslFramelength>4096)
                DebugBreak()

        }

        void Init()
        {
            pOriginalSamples = 0
            pDistortedSamples = 0
            pIgnoreFrameFlags = 0
            pActiveFrameFlags = 0
            pAslActiveFrameFlags = 0
            pAslActiveFrameFlagsDeg = 0
            pStartSampleUtterance = 0
            pStopSampleUtterance = 0
            pDelayUtterance = 0
            pDelayReliability = 0
            pPitchVecOfRef = 0
            pPitchVecOfDeg = 0
        }

        void Free()
```

```
135             {
136                 if(pIgnoreFrameFlags) matFree(pIgnoreFrameFlags)
137                 if(pActiveFrameFlags) matFree(pActiveFrameFlags)
138                 if (pAslActiveFrameFlags) matFree(pAslActiveFrameFlags)
139                 if (pAslActiveFrameFlagsDeg) matFree(pAslActiveFrameFlagsDeg)
140                 if (pDelayReliability) matFree(pDelayReliability)
141                 if(pPitchVecOfRef) matFree(pPitchVecOfRef)
142                 if(pPitchVecOfDeg) matFree(pPitchVecOfDeg)
143                 Init()
144             }
145
146             void AllocVectors(int NumFrames)
147             {
148                 MaxModelFramesAlloc = NumFrames
149                 pIgnoreFrameFlags = (int*) matMalloc(sizeof(int)*NumFrames)
150                 pActiveFrameFlags = (bool*) matMalloc(sizeof(bool)*NumFrames)
151                 pAslActiveFrameFlags = (bool*)matMalloc(sizeof(bool)*NumFrames)
152                 pAslActiveFrameFlagsDeg = (bool*)matMalloc(sizeof(bool)*NumFrames)
153                 pDelayReliability = (XFLOAT*)matMalloc(sizeof(XFLOAT)*NumFrames)
154                 pPitchVecOfRef = (XFLOAT*) matMalloc(sizeof(XFLOAT)*NumFrames)
155                 pPitchVecOfDeg = (XFLOAT*) matMalloc(sizeof(XFLOAT)*NumFrames)
156                 MaxModelFrames = NumFrames
157
158                 for (int i=0  i<NumFrames  i++) pActiveFrameFlags[i] = false
159                 for (int i=0  i<NumFrames  i++) pAslActiveFrameFlags[i] = false
160                 for (int i=0  i<NumFrames  i++) pAslActiveFrameFlagsDeg[i] = false
161                 matbSet(0.0, pPitchVecOfRef, NumFrames)
162                 matbSet(0.0, pPitchVecOfDeg, NumFrames)
163                 matbSet(0.0, pDelayReliability, NumFrames)
164             }
165
166             void SetUtteranceInfo(int NumUtterances, int* pStart, int* pStop, int* pDelay)
167             {
168                 pStartSampleUtterance->SetSize(NumUtterances)
169                 pStopSampleUtterance->SetSize(NumUtterances)
170                 pDelayUtterance->SetSize(NumUtterances)
171                 for (int f=0  f < NumUtterances  f++)
172                 {
173                     (pStartSampleUtterance->m_pData)[f] = pStart[f]
174                     (pStopSampleUtterance->m_pData)[f] = pStop[f]
175                     (pDelayUtterance->m_pData)[f] = pDelay[f]
176                 }
177             }
178
179             void Print(FILE* pFile)
180
181             int Framesize
182             int MaxModelFrames
183             int MaxModelFramesAlloc
184             int FramesUsed
185             long MaxSigLen
186             FILE* LogFile
187
188             long            OriginalNumberOfSamples
189             XFLOAT*         pOriginalSamples
190
191             long            DistortedNumberOfSamples
192             XFLOAT*         pDistortedSamples
193
194             CIntArray*      pStartSampleUtterance
195             CIntArray*      pStopSampleUtterance
196             CIntArray*      pDelayUtterance
197             int             FirstRefSample
198             int             FirstDegSample
199
200             XFLOAT* pDelayReliability
201
202             XFLOAT pNoiseDuringSpeechdB[2]
```

```
            XFLOAT pNoiseDuringSilencedB[2]
            XFLOAT pBGNSwitchingLevel[2]

            int PitchFrameSize
            XFLOAT PitchFreqRef
            XFLOAT PitchFreqDeg
            XFLOAT* pPitchVecOfRef
            XFLOAT* pPitchVecOfDeg

            bool*           pActiveFrameFlags

            int*            pIgnoreFrameFlags

            int             AslFramelength
            int             AslFrames
            bool*           pAslActiveFrameFlags
            int             AslFramesDeg
            bool*           pAslActiveFrameFlagsDeg

            MAT_HANDLE mh
    }

    bool DoCalculateDelayDegPlus(CDelayPara* pDelayPara, POLQA_RESULT_DATA* PolqaResults)
}


{

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:
```

```
339         CParameters()
340         {
341             int i
342             mTAPara.mCorrForSkippingInitialDelaySearch = 0.6F
343             mTAPara.CoarseAlignSegmentLengthInMs = 600
344 
345             SPEECH_WINDOW_PARA      SpeechWinPara[] =
346             {
347                     {8000,   32, 32,
348                         {128,  256, 128,  64,  32,  0, 0},
349                         {-1,    -1,  -1,  86,  34,  0,  0},
350                         {-1,    -1,  -1,  15,  12,  0,  0}},
351                     {16000,  64, 64,
352                         {256,  512, 256, 128,  64,  0},
353                         {-1,    -1,  -1,  63,  33,  0},
354                         {-1,    -1,  -1,  13,  10,  0}},
355                     {48000, 256, 256,
356                         {512, 1024, 512, 512, 128,  0},
357                         {-1,    -1,  -1, 115,  61,  0},
358                         {-1,    -1,  -1,  17,  16,  0}}
359             }
360 
361             for (i=0  i<3  i++)
362             {
363                 mSpeechTAPara.Win[i].Samplerate = SpeechWinPara[i].Samplerate
364                 mSpeechTAPara.Win[i].mDelayFineAlignCorrlen =
    SpeechWinPara[i].mDelayFineAlignCorrlen
365                 mSpeechTAPara.Win[i].mSRDetectFineAlignCorrlen =
    SpeechWinPara[i].mSRDetectFineAlignCorrlen
366                 for (int k=0  k<8  k++)
367                 {
368                     mSpeechTAPara.Win[i].CoarseAlignCorrlen[k] =
    SpeechWinPara[i].CoarseAlignCorrlen[k]
369                     mSpeechTAPara.Win[i].WindowSize[k]         = SpeechWinPara[i].WindowSize[k]

370                     mSpeechTAPara.Win[i].pViterbiDistanceWeightFactor[k] =
    SpeechWinPara[i].pViterbiDistanceWeightFactor[k]
371                 }
372             }
373             mSpeechTAPara.LowEnergyThresholdFactor = 15.0F
374             mSpeechTAPara.LowCorrelThreshold = 0.4F
375             mSpeechTAPara.FineAlignLowEnergyThresh = 2.0
376             mSpeechTAPara.FineAlignLowEnergyCorrel = 0.6F
377             mSpeechTAPara.FineAlignShortDropOfCorrelR = -1
378             mSpeechTAPara.FineAlignShortDropOfCorrelRLastBest = 0.65F
379 
380             mSpeechTAPara.ViterbiDistanceWeightFactorDist = 5
381 
382             SPEECH_WINDOW_PARA      AudioWinPara[] =
383             {
384                     {8000,   32, 32,
385                         {64,   128,  64,  64,  16,  0, 0},
386                         {-1,    -1,  -1, 128,  32,  0, 0},
387                         {-1,    -1,  -1,   6,   6,  0, 0}},
388                     {16000,  64, 64,
389                         {128,  256, 128, 128,  32,  0},
390                         {-1,    -1,  -1,  64,  32,  0},
391                         {-1,    -1,  -1,  12,  12,  0}},
392                     {48000, 256, 2048,
393                         {512, 1024, 512, 512, 256,  128,   0},
394                         {-1,    -1,  -1, 512, 1024, 2048,  0},
395                         {-1,    -1,  -1,  16,  16,  32,    0}}
396             }
397 
398             for (i=0  i<3  i++)
399             {
400                 mAudioTAPara.Win[i].Samplerate = AudioWinPara[i].Samplerate
401                 mAudioTAPara.Win[i].mDelayFineAlignCorrlen    =
```

```
    AudioWinPara[i].mDelayFineAlignCorrlen
402                 mAudioTAPara.Win[i].mSRDetectFineAlignCorrlen =
    AudioWinPara[i].mSRDetectFineAlignCorrlen
403                 for (int k=0  k<8  k++)
404                 {
405                     mAudioTAPara.Win[i].CoarseAlignCorrlen[k] =
    AudioWinPara[i].CoarseAlignCorrlen[k]
406                     mAudioTAPara.Win[i].WindowSize[k]         = AudioWinPara[i].WindowSize[k]
407                     mAudioTAPara.Win[i].pViterbiDistanceWeightFactor[k] =
    AudioWinPara[i].pViterbiDistanceWeightFactor[k]
408                 }
409             }
410             mAudioTAPara.LowEnergyThresholdFactor = 1
411             mAudioTAPara.LowCorrelThreshold = 0.85F
412             mAudioTAPara.FineAlignLowEnergyThresh = 32.0
413             mAudioTAPara.FineAlignLowEnergyCorrel = 0.8F
414             mAudioTAPara.FineAlignShortDropOfCorrelR = -1
415             mAudioTAPara.FineAlignShortDropOfCorrelRLastBest = 0.8F
416             mAudioTAPara.ViterbiDistanceWeightFactorDist = 6
417
418             mSREPara.LowEnergyThresholdFactor = 15.0F
419             mSREPara.LowCorrelThreshold = 0.4F
420
421             mSREPara.MaxStepPerFrame = 160
422             mSREPara.MaxBins = ((int)(mSREPara.MaxStepPerFrame*2.0*0.9))
423
424             mSREPara.MaxWin=4
425             mSREPara.LowPeakEliminationThreshold=0.2000000029802322F
426             mSREPara.PercentageRequired = 0.04F
427
428             mSREPara.LargeStepLimit = 0.08F
429             mSREPara.MaxDistanceToLast = 7
430             mSREPara.MaxLargeStep = 5
431             mSREPara.MaxDistance = 14
432
433             mSREPara.MinReliability = 7
434             mSREPara.MinFrequencyOfOccurrence = 3
435
436             mSREPara.AllowedDistancePara2 = 0.85F
437             mSREPara.AllowedDistancePara3 = 1.5F
438
439             mSREPara.ReliabilityThreshold = 0.3F
440             mSREPara.MinHistogramData = 8
441
442             mViterbi.UseRelDistance = false
443             mViterbi.FrameWeightWeight = 1.0F
444         }
445
446         void Init(long Samplerate)
447         {
448             mSREPara.Init(Samplerate)
449         }
450
451         VITERBI_PARA        mViterbi
452         GENERAL_TA_PARA     mTAPara
453         SPEECH_TA_PARA      mSpeechTAPara
454         AUDIO_TA_PARA       mAudioTAPara
455         SR_ESTIMATION_PARA  mSREPara
456 }
457 }
458
459
460 {
461
462 class CProcessData
463 {
464     public:
465         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

            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
        {
            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)
```

```
806         mpAslActiveFrameFlags = NULL
807         matFree(mpAslActiveFrameFlagsDeg)
808         mpAslActiveFrameFlagsDeg = NULL
809     }
810 
811     long mNumFrames
812     int mStepsize
813     int mResolutionInSamples
814     int mPitchFrameSize
815     long *mpDelay
816     OTA_FLOAT *mpReliability
817     OTA_FLOAT mAvgReliability
818     OTA_FLOAT mRelSamplerateDev
819 
820     int  mNumUtterances
821     int* mpStartSampleUtterance
822     int* mpStopSampleUtterance
823     int* mpDelayUtterance
824     int FirstRefSample
825     int FirstDegSample
826 
827     int         mNumSections
828     SECTION     *mpRefSections
829     SECTION     *mpDegSections
830 
831     double mSNRRefdB, mSNRDegdB
832     double mNoiseLevelRef, mNoiseLevelDeg
833     double mSignalLevelRef, mSignalLevelDeg
834     double mNoiseThresholdRef, mNoiseThresholdDeg
835 
836     int *mpActiveFrameFlags
837 
838     int *mpIgnoreFlags
839     int mNumIngoreFlags
840     int mAslFrames
841     int mAslFramelength
842     int *mpAslActiveFrameFlags
843     int mAslFramesDeg
844     int *mpAslActiveFrameFlagsDeg
845 
846     double mTimeDiffs[5]
847 
848 }OTA_RESULT
849 
850 struct FilteringParameters
851 {
852     int pListeningCondition
853     double cutOffFrequencyLow
854     double cutOffFrequencyHigh
855     double disturbedEnergyQuotient
856 }
857 
858 class ITempAlignment
859 {
860     public:
861 
862         virtual bool Init(CProcessData* pProcessData)=0
863         virtual void Free()=0
864         virtual void Destroy()=0
865 
866         virtual bool SetSignal(int Index, unsigned long SampleRate, unsigned long NumSamples, int
    NumChannels, OTA_FLOAT** pSignal)=0
867 
868         virtual void GetFilterCharacteristics(FilteringParameters *FilterParams)=0
869 
870         virtual bool FilterSignal(int Index, FilteringParameters *FilterParams)=0
871 
872         virtual bool Run(unsigned long Control, OTA_RESULT* pResult, int TArunIndex)=0
```

```
873
874         virtual void GetNoiseSwitching(OTA_FLOAT* pBGNSwitchingLevel, OTA_FLOAT*
    pNoiseLevelSpeechDeg, OTA_FLOAT* pNoiseLevelSilenceDeg)=0
875
876         virtual OTA_FLOAT GetPitchFreq(int Signal, int Channel)=0
877
878         virtual OTA_FLOAT GetPitchVector(int Signal, int Channel, OTA_FLOAT* pVector, int NumFrames,
    int SamplesPerFrame)=0
879         virtual int GetPitchFrameSize()=0
880 }
881
882 enum AlignmentType
883 {
884     TA_FOR_SPEECH=0,
885
886 }
887
888 ITempAlignment* CreateAlignment(AlignmentType Type)
889
890 }
891
892
893 {
894
895 extern XFLOAT aCmdlineArray[]
896 extern int    aElementsInCmdlineArray
897
898 void DeleteUtteranceFromVector(int UttIndexToDelete, CDelayPara* pDelayPara)
899 {
900     for (int i=UttIndexToDelete  i< pDelayPara->pDelayUtterance->GetSize()-1  i++)
901     {
902         pDelayPara->pDelayUtterance->m_pData[i] = pDelayPara->pDelayUtterance->m_pData[i+1]
903         pDelayPara->pStartSampleUtterance->m_pData[i] =
    pDelayPara->pStartSampleUtterance->m_pData[i+1]
904         pDelayPara->pStopSampleUtterance->m_pData[i] =
    pDelayPara->pStopSampleUtterance->m_pData[i+1]
905     }
906     pDelayPara->pDelayUtterance->SetSize(pDelayPara->pDelayUtterance->GetSize()-1)
907     pDelayPara->pStartSampleUtterance->SetSize(pDelayPara->pStartSampleUtterance->GetSize()-1)
908     pDelayPara->pStopSampleUtterance->SetSize(pDelayPara->pStopSampleUtterance->GetSize()-1)
909 }
910
911 XFLOAT GetNearestPitch(XFLOAT Target, XFLOAT a, XFLOAT b, XFLOAT c)
912 {
913     if (Target<0.6*a && Target>0.4*a) a /= 2.0
914     if (Target<0.6*b && Target>0.4*b) b /= 2.0
915     if (Target<0.6*c && Target>0.4*c) c /= 2.0
916     if (a<0.6*Target && a>0.4*Target) Target /= 2.0
917
918     XFLOAT Res = a
919     if (abs(Target-Res)>abs(Target-b)) Res = b
920     if (abs(Target-Res)>abs(Target-c)) Res = c
921     return Res
922 }
923
924 void CorrectPitchVectors(CDelayPara* pDelayPara)
925 {
926     int NumFrames = pDelayPara->MaxModelFrames
927     for (int i=0  i<NumFrames  i++)
928     {
929         if (pDelayPara->pPitchVecOfRef[i] && pDelayPara->pPitchVecOfDeg[i])
930         {
931             if (pDelayPara->pPitchVecOfRef[i] / pDelayPara->pPitchVecOfDeg[i] > 0.4 &&
    pDelayPara->pPitchVecOfRef[i] / pDelayPara->pPitchVecOfDeg[i] <0.6)
932                 pDelayPara->pPitchVecOfDeg[i] /= 2.0
933             if (pDelayPara->pPitchVecOfRef[i] / pDelayPara->pPitchVecOfDeg[i] > 1.8 &&
    pDelayPara->pPitchVecOfRef[i] / pDelayPara->pPitchVecOfDeg[i] <2.2)
934                 pDelayPara->pPitchVecOfRef[i] /= 2.0
```

```
        }
    }
}

void AlignPitchVectors(CDelayPara* pDelayPara, XFLOAT* pPitchVec, XFLOAT* pTarget, int* IgnoreFlags)
{
    int i, frameIndex
    XFLOAT OverlapCoeff = (1 - 0.75)

    int FrameLength = pDelayPara->Framesize
    int NumFrames = pDelayPara->MaxModelFrames
    XFLOAT *pPitchVecAligned = (XFLOAT*)matMalloc(NumFrames * sizeof(XFLOAT))
    for (frameIndex = 0  frameIndex<NumFrames  frameIndex++)
    {
        pPitchVecAligned[frameIndex] = 0

        if (IgnoreFlags[frameIndex]!=0x0004 && IgnoreFlags[frameIndex]!=0x0008)

        {
            int utt = GetUtteranceForFrame(*pDelayPara->pStartSampleUtterance,
*pDelayPara->pStopSampleUtterance, *pDelayPara->pDelayUtterance, frameIndex, FrameLength
/ (1 - 0.75))
            if (utt>=0)
            {
                ASSERT(utt<pDelayPara->pDelayUtterance->GetSize())
                int DegFrame = (*pDelayPara->pDelayUtterance).m_pData[utt]
                DegFrame = frameIndex+(DegFrame+FrameLength*OverlapCoeff)/FrameLength
                if (DegFrame>=0 && DegFrame<NumFrames)
                {
                    if (DegFrame>0 && DegFrame<NumFrames-2)
                    {
                        pPitchVecAligned[frameIndex] = GetNearestPitch(pTarget[frameIndex],
pPitchVec[DegFrame], pPitchVec[DegFrame+1], pPitchVec[DegFrame-1])
                    }
                    else
                        pPitchVecAligned[frameIndex] = pPitchVec[DegFrame]
                }
                else
                    pPitchVecAligned[frameIndex] = 0
            }
        }
    }

    for (i=0  i<NumFrames  i++)
        pPitchVec[i] = pPitchVecAligned[i]

    CorrectPitchVectors(pDelayPara)

    matFree(pPitchVecAligned)

}

XFLOAT AveragePitchFrequencyFromPitchVector(CDelayPara* pDelayPara, XFLOAT* pPitchVec)
{
    const XFLOAT BinWidth = 1.0
    const int MaxBins = (int)(500.0 / BinWidth)
    int NumFrames = pDelayPara->MaxModelFrames
    if (NumFrames<=0) OPTTHROW(CALCULATION_FAILED)

    XFLOAT MaxPitch = matMax(pPitchVec, NumFrames)
    XFLOAT MinPitch = MaxPitch
    for (int i=0  i<NumFrames  i++)
        if (pPitchVec[i]>0 && pPitchVec[i]<MinPitch)
            MinPitch = pPitchVec[i]

    if (MaxPitch>30000) OPTTHROW(CALCULATION_FAILED)
    if (MaxPitch<0) OPTTHROW(CALCULATION_FAILED)
    if (MinPitch>30000) OPTTHROW(CALCULATION_FAILED)
```

```
    if (MinPitch<0) OPTTHROW(CALCULATION_FAILED)

    if (MaxPitch-MinPitch<5*BinWidth)
    {
        MinPitch -= 3*BinWidth
        MaxPitch += 3*BinWidth
    }

    int NumBins = (int)((MaxPitch-MinPitch)/BinWidth+0.5)
    if (NumBins>MaxBins) NumBins = MaxBins
    if (NumBins<=0)
    {
        printf("\nNumBins=%d, BinWidth=%.1f, MinPitch=%.1f, MaxPitch=%.1f\n", NumBins, BinWidth,
MinPitch, MaxPitch)  OPTTHROW(CALCULATION_FAILED)
    }
    if (NumBins>10000) OPTTHROW(CALCULATION_FAILED)

    XFLOAT *pPitchHistogram = (XFLOAT*)matMalloc(NumBins * sizeof(XFLOAT))
    if (!pPitchHistogram) OPTTHROW(CALCULATION_FAILED)

    matbSet(0.0, pPitchHistogram, NumBins)
    for (int i=0  i<NumFrames  i++)
    {
        if (pPitchVec[i]>0.0)
        {
            int Bin = (((0.0) > ((((NumBins-1) < ((pPitchVec[i]-MinPitch) / BinWidth + 0.5)) ?
(NumBins-1) : ((pPitchVec[i]-MinPitch) / BinWidth + 0.5)))) ? (0.0) : ((((NumBins-1) <
((pPitchVec[i]-MinPitch) / BinWidth + 0.5)) ? (NumBins-1) : ((pPitchVec[i]-MinPitch) /
BinWidth + 0.5))))
            pPitchHistogram[Bin]++
        }
    }

    int MaxPos
    matMaxExt(pPitchHistogram, NumBins, &MaxPos)

    int MinBin = (((0) > (MaxPos - 30.0 / BinWidth + 0.25)) ? (0) : (MaxPos - 30.0 / BinWidth +
0.25))
    int MaxBin = (((NumBins) < (MaxPos + 30.0 / BinWidth + 0.25)) ? (NumBins) : (MaxPos + 30.0 /
BinWidth + 0.25))
    XFLOAT Avg=0.0
    int Num = 0
    for (int i=MinBin  i<MaxBin  i++)
    { Avg+=i*BinWidth*pPitchHistogram[i]  Num+=pPitchHistogram[i] }
    if (Num>0)
    {
        Avg /= Num
        Avg += MinPitch
    }
    else Avg = 0

      matFree(pPitchHistogram)

      return Avg
}

template <class T>
int ConvertFrameRate(int NumFramesIn, T const *pFramesIn, int FrameSizeIn, int NumFramesOut, T*
pFramesOut, int FrameSizeOut)
{
    if (FrameSizeOut==FrameSizeIn)
    {
        memcpy(pFramesOut, pFramesIn, sizeof(T)*(((NumFramesIn) < (NumFramesOut)) ? (NumFramesIn) :
(NumFramesOut)))
        for (int i=NumFramesIn  i<NumFramesOut  i++)
            pFramesOut[i] = pFramesIn[NumFramesIn-1]
        NumFramesOut = NumFramesIn
    }
```

```
    else if (FrameSizeOut>FrameSizeIn)
    {
        int NextFramePosIn=FrameSizeIn>>1
        int NextFrameIn=0
        int NextFramePosOut=0
        int i
        for (i=0  i<NumFramesOut  i++)
        {
            while (NextFrameIn<NumFramesIn-1 && NextFramePosIn<NextFramePosOut)
            { NextFramePosIn+=FrameSizeIn  NextFrameIn++ }

            pFramesOut[i] = pFramesIn[NextFrameIn]
            NextFramePosOut += FrameSizeOut
        }
        NumFramesOut = i
    }
    else if (FrameSizeOut<FrameSizeIn)
    {
        int NextFramePosIn=0
        int NextFrameIn=0
        int NextFramePosOut=FrameSizeOut>>1
        int i
        for (i=0  i<NumFramesOut  i++)
        {
            while (NextFrameIn<NumFramesIn-1 && NextFramePosIn<NextFramePosOut)
            { NextFramePosIn+=FrameSizeIn  NextFrameIn++ }

            pFramesOut[i] = pFramesIn[NextFrameIn]
            NextFramePosOut += FrameSizeOut
        }
        NumFramesOut = i
    }
    return NumFramesOut
}

void ConvertFrameRateOfVectors(OTA_RESULT* pTaResult, CDelayPara* pDelayPara, POLQA_RESULT_DATA*
PolqaResults)
{
    pDelayPara->FramesUsed=ConvertFrameRate(pTaResult->mNumFrames, pTaResult->mpReliability,
pTaResult->mStepsize, pDelayPara->MaxModelFrames, PolqaResults->m_DelayReliabilityPerFrame,
pDelayPara->Framesize)
    pDelayPara->FramesUsed=ConvertFrameRate(pTaResult->mNumFrames, pTaResult->mpDelay,
pTaResult->mStepsize, pDelayPara->MaxModelFrames, PolqaResults->m_DelayPerFrame,
pDelayPara->Framesize)
}

int ConvertActivityFlag(FILE*  pLogFile, OTA_RESULT* pTaResult, CDelayPara* pDelayPara,
POLQA_RESULT_DATA* PolqaResults)
{
    float modelToMacroSize = (float)pDelayPara->Framesize/(float)pTaResult->mStepsize

    int macroFrame
    for(int fr = 0  fr < pDelayPara->MaxModelFrames  fr++)
    {
        macroFrame = (int)(modelToMacroSize * fr + 0.5)
        if (macroFrame < pTaResult->mNumFrames)
        {
            pDelayPara->pActiveFrameFlags[fr] = pTaResult->mpActiveFrameFlags[macroFrame] == 1
            pDelayPara->pIgnoreFrameFlags[fr] = pTaResult->mpIgnoreFlags[macroFrame]
        }
        else
        {
            pDelayPara->pActiveFrameFlags[fr] = false
            pDelayPara->pIgnoreFrameFlags[fr] = 0
        }
    }
    return (int)((pTaResult->mNumFrames-0.5)/modelToMacroSize)
}
```

```
1122
1123 void
1124
1125 bool DoAlignmentPlus(CDelayPara* pDelayPara, POLQA_RESULT_DATA* PolqaResults, ITempAlignment* pTA,
     OTA_RESULT** pTaResult, unsigned long Mode, bool* pDone)
1126 {
1127     bool rc = true
1128     long f
1129     long ClockCycles
1130     double TimeDiff
1131     MAT_HANDLE mh = pDelayPara->mh
1132     TACheckTimeMatInit(mh, 1)
1133
1134     CProcessData TAInitData
1135     TAInitData.mpMathlibHandle = pDelayPara->mh
1136     TAInitData.mpLogFile = pDelayPara->LogFile
1137     TAInitData.mSamplerate = PolqaResults->m_SampleFrequencyHz
1138     TAInitData.mEnablePlotting = 0
1139     TAInitData.mStartPlotIteration = -100
1140     TAInitData.mLastPlotIteration = -10
1141     TAInitData.mMinLowVarDelayInSamples  = -(int(0.3*PolqaResults->m_SampleFrequencyHz))
1142     TAInitData.mMaxHighVarDelayInSamples = (int(0.3*PolqaResults->m_SampleFrequencyHz))
1143
1144     int const MAX_TA_RUNS = 2
1145
1146     int   TArunIndex        = -1
1147     int   bestAlignmentIdx  = 0
1148     float maxAvgReliability = -1.0f
1149     OTA_RESULT *pResamplingResults[MAX_TA_RUNS]
1150     for (int i = 0  i < MAX_TA_RUNS  i++)
1151         pResamplingResults[i] = new OTA_RESULT()
1152     PolqaResults->m_ResamplingApplied = false
1153     CDelayPara  pDelayParaSaved[MAX_TA_RUNS]
1154
1155     XFLOAT *pRefSig[2] = {NULL, NULL}, *pDegSig[2] = {NULL, NULL}
1156     unsigned long NewLen
1157     long numRefSamples = pDelayPara->OriginalNumberOfSamples, numDegSamples =
     pDelayPara->DistortedNumberOfSamples
1158     pRefSig[0] = (XFLOAT*)matMalloc(numRefSamples * sizeof(XFLOAT))
1159
1160     pDegSig[0] = (XFLOAT*)matMalloc(numDegSamples * sizeof(XFLOAT))
1161
1162     matbCopy(pDelayPara->pOriginalSamples,  pRefSig[0], numRefSamples)
1163     matbCopy(pDelayPara->pDistortedSamples, pDegSig[0], numDegSamples)
1164
1165     bool Done = false
1166     do
1167     {
1168         pTA->Init(&TAInitData)
1169
1170         pTA->SetSignal(0, PolqaResults->m_SampleFrequencyHz, pDelayPara->OriginalNumberOfSamples, 1,
     pRefSig)
1171
1172         pTA->SetSignal(1, PolqaResults->m_SampleFrequencyHz, pDelayPara->DistortedNumberOfSamples,
     1, pDegSig)
1173
1174         FilteringParameters FilterParas
1175         FilterParas.pListeningCondition = PolqaResults->m_ListeningCondition
1176         FilterParas.cutOffFrequencyLow  = 3200
1177         FilterParas.cutOffFrequencyHigh = 3400
1178
1179         pTA->FilterSignal(1, &FilterParas)
1180         pTA->FilterSignal(0, &FilterParas)
1181
1182         TACheckTimeMatEval(mh, 1, &ClockCycles, &TimeDiff)
1183         if (TAInitData.mpLogFile)
1184             fprintf(TAInitData.mpLogFile, "\nTime required to set up the temporal alignment:
     %.3lfs\n", TimeDiff)
```

```
1185
1186        AddProcessingTime(PolqaResults, "TA Initialization", TimeDiff, ClockCycles)
1187
1188        TACheckTimeMatInit(mh, 1)
1189        TArunIndex++
1190
1191        pResamplingResults[TArunIndex]->mAslFramelength = pDelayPara->Framesize
1192        pResamplingResults[TArunIndex]->mAslFrames = pDelayPara->MaxModelFrames
1193        pResamplingResults[TArunIndex]->mAslFramesDeg = pDelayPara->MaxModelFrames
1194
1195        rc = pTA->Run(Mode | 0x2 | 0x1, pResamplingResults[TArunIndex], TArunIndex)
1196
1197
1198
1199        if (pResamplingResults[TArunIndex]->mNumFrames>pDelayPara->MaxModelFrames)
1200            DebugBreak()
1201
1202        if (rc)
1203        {
1204            pDelayParaSaved[TArunIndex].LogFile = pDelayPara->LogFile
1205            pDelayParaSaved[TArunIndex].mh = pDelayPara->mh
1206            pDelayParaSaved[TArunIndex].MaxSigLen = pDelayPara->MaxSigLen
1207            pDelayParaSaved[TArunIndex].pStartSampleUtterance = pDelayPara->pStartSampleUtterance
1208            pDelayParaSaved[TArunIndex].pStopSampleUtterance = pDelayPara->pStopSampleUtterance
1209            pDelayParaSaved[TArunIndex].pDelayUtterance = pDelayPara->pDelayUtterance
1210            pDelayParaSaved[TArunIndex].pOriginalSamples = pDelayPara->pOriginalSamples
1211            pDelayParaSaved[TArunIndex].OriginalNumberOfSamples =
    pDelayPara->OriginalNumberOfSamples
1212            pDelayParaSaved[TArunIndex].pDistortedSamples = pDelayPara->pDistortedSamples
1213            pDelayParaSaved[TArunIndex].DistortedNumberOfSamples =
    pDelayPara->DistortedNumberOfSamples
1214            pDelayParaSaved[TArunIndex].Framesize = pDelayPara->Framesize
1215
1216            pDelayParaSaved[TArunIndex].AllocVectors(pDelayPara->MaxModelFramesAlloc)
1217            pDelayParaSaved[TArunIndex].MaxModelFrames = pDelayPara->MaxModelFrames
1218            pDelayParaSaved[TArunIndex].FramesUsed = (((pDelayPara->MaxModelFrames) <
    (pResamplingResults[TArunIndex]->mNumFrames)) ? (pDelayPara->MaxModelFrames) :
    (pResamplingResults[TArunIndex]->mNumFrames))
1219
1220            pDelayParaSaved[TArunIndex].AslFramelength =
    pResamplingResults[TArunIndex]->mAslFramelength
1221            pDelayParaSaved[TArunIndex].AslFrames = pResamplingResults[TArunIndex]->mAslFrames
1222            pDelayParaSaved[TArunIndex].AslFramesDeg = pResamplingResults[TArunIndex]->mAslFramesDeg
1223            pDelayParaSaved[TArunIndex].FramesUsed = pResamplingResults[TArunIndex]->mNumFrames
1224            for(int i=0  i<(((pDelayParaSaved[TArunIndex].AslFrames) <
    (pDelayParaSaved[TArunIndex].MaxModelFrames)) ? (pDelayParaSaved[TArunIndex].AslFrames)
    : (pDelayParaSaved[TArunIndex].MaxModelFrames))  i++)
1225                pDelayParaSaved[TArunIndex].pAslActiveFrameFlags[i] =
    (pResamplingResults[TArunIndex]->mpAslActiveFrameFlags[i] != 0)
1226
1227            for (int i=0  i<(((pDelayParaSaved[TArunIndex].AslFramesDeg) <
    (pDelayParaSaved[TArunIndex].MaxModelFrames)) ?
    (pDelayParaSaved[TArunIndex].AslFramesDeg) :
    (pDelayParaSaved[TArunIndex].MaxModelFrames))  i++)
1228                pDelayParaSaved[TArunIndex].pAslActiveFrameFlagsDeg[i] =
    (pResamplingResults[TArunIndex]->mpAslActiveFrameFlagsDeg[i] != 0)
1229
1230            for(int i=0  i<pDelayParaSaved[TArunIndex].FramesUsed  i++)
1231                pDelayParaSaved[TArunIndex].pActiveFrameFlags[i] =
    pResamplingResults[TArunIndex]->mpActiveFrameFlags[i]
1232
1233            for(int i=0  i<pDelayParaSaved[TArunIndex].FramesUsed  i++)
1234                pDelayParaSaved[TArunIndex].pIgnoreFrameFlags[i] =
    pResamplingResults[TArunIndex]->mpIgnoreFlags[i]
1235
1236            if (pResamplingResults[TArunIndex]->mpReliability)
1237                matbCopy(pResamplingResults[TArunIndex]->mpReliability,
    pDelayParaSaved[TArunIndex].pDelayReliability,
```

```
    pDelayParaSaved[TArunIndex].FramesUsed)
1238
1239
1240            pTA->GetNoiseSwitching(pDelayParaSaved[TArunIndex].pBGNSwitchingLevel,
    pDelayParaSaved[TArunIndex].pNoiseDuringSpeechdB,
    pDelayParaSaved[TArunIndex].pNoiseDuringSilencedB)
1241
1242            pDelayParaSaved[TArunIndex].FirstRefSample =
    pResamplingResults[TArunIndex]->FirstRefSample
1243            pDelayParaSaved[TArunIndex].FirstDegSample =
    pResamplingResults[TArunIndex]->FirstDegSample
1244
1245            {
1246
1247                TACheckTimeMatInit(mh, 1)
1248                pDelayParaSaved[TArunIndex].PitchFreqRef = pTA->GetPitchVector(0, 0,
    pDelayParaSaved[TArunIndex].pPitchVecOfRef,
    pDelayParaSaved[TArunIndex].MaxModelFrames, pDelayParaSaved[TArunIndex].Framesize)
1249                pDelayParaSaved[TArunIndex].PitchFreqDeg = pTA->GetPitchVector(1, 0,
    pDelayParaSaved[TArunIndex].pPitchVecOfDeg,
    pDelayParaSaved[TArunIndex].MaxModelFrames, pDelayParaSaved[TArunIndex].Framesize)
1250                pDelayParaSaved[TArunIndex].PitchFrameSize = pTA->GetPitchFrameSize()
1251                TACheckTimeMatEval(mh, 1, &ClockCycles, &TimeDiff)
1252                if (pDelayPara->LogFile)
1253                    fprintf(pDelayPara->LogFile, "\nTime required to calculate pitch information:
    %.3lfs\n", TimeDiff)
1254            }
1255
1256            TACheckTimeMatEval(mh, 1, &ClockCycles, &TimeDiff)
1257            if (TAInitData.mpLogFile)
1258                fprintf(TAInitData.mpLogFile, "\nTime spent in pTA->Run() function: %.3lfs\n",
    TimeDiff)
1259            TACheckTimeMatInit(mh, 1)
1260            AddProcessingTime(PolqaResults, "TA Run", TimeDiff, ClockCycles)
1261
1262            AddProcessingTime(PolqaResults, "TA BeforeInitalDelay",
    pResamplingResults[TArunIndex]->mTimeDiffs[0], 0)
1263            AddProcessingTime(PolqaResults, "TA InitalDelay",
    pResamplingResults[TArunIndex]->mTimeDiffs[1], 0)
1264            AddProcessingTime(PolqaResults, "TA CoarseAlignment",
    pResamplingResults[TArunIndex]->mTimeDiffs[2], 0)
1265            AddProcessingTime(PolqaResults, "TA FineAlignment",
    pResamplingResults[TArunIndex]->mTimeDiffs[3], 0)
1266
1267            OTA_FLOAT SrcThreshold = 0.005
1268            if (1 && TArunIndex < MAX_TA_RUNS-1 && pResamplingResults[TArunIndex]->mRelSamplerateDev
    > 0 && (pResamplingResults[TArunIndex]->mRelSamplerateDev > 1.0 + SrcThreshold ||
    pResamplingResults[TArunIndex]->mRelSamplerateDev < 1.0 - SrcThreshold))
1269            {
1270
1271
1272                    PolqaResults->m_ResamplingApplied = true
1273
1274                    XFLOAT Ratio = 1.0 + (1.0 - pResamplingResults[TArunIndex]->mRelSamplerateDev)
1275
1276                    if (Ratio<1.0)
1277                    {
1278                        if (TAInitData.mpLogFile)
1279                            fprintf(TAInitData.mpLogFile, "\n\n***** Downsampling degraded signal by
    %.3lf%%\n\n", (float)(100.0*(1.0-Ratio)))
1280
1281                        matConvertSamplerate(pDegSig[0], numDegSamples, pDegSig[0],
    pDelayPara->MaxSigLen, Ratio, &NewLen)
1282
1283                        numDegSamples = NewLen
1284                    }
1285                    else
1286                    {
```

```
                        if (TAInitData.mpLogFile)
                            fprintf(TAInitData.mpLogFile, "\n\n***** Downsampling reference signal by %.3lf%%\n\n", (float)(100.0*(Ratio-1.0)))

                        matConvertSamplerate(pRefSig[0], numRefSamples, pRefSig[0], pDelayPara->MaxSigLen, 1.0/Ratio, &NewLen)

                        numRefSamples = NewLen
                    }

            }
            else
            {
                Done = true
            }
        }
    }   while (!Done && rc)

    if (rc)
    {
        for (int i = 0  i <= TArunIndex  i++)


        if (TArunIndex==0)
        {
            bestAlignmentIdx = 0
            maxAvgReliability  = pResamplingResults[0]->mAvgReliability
        }
        else
        {

            if (
                (    fabs(1.0 - pResamplingResults[1]->mRelSamplerateDev) < (fabs(1.0 - pResamplingResults[0]->mRelSamplerateDev)) ||
(pResamplingResults[1]->mRelSamplerateDev == -1.0)    )
                && (pResamplingResults[1]->mAvgReliability > pResamplingResults[0]->mAvgReliability)
               )

            {
                bestAlignmentIdx = 1
                maxAvgReliability  = pResamplingResults[1]->mAvgReliability
            }
            else
            {
                bestAlignmentIdx = 0
                maxAvgReliability  = pResamplingResults[0]->mAvgReliability
            }
        }



        *pDelayPara = pDelayParaSaved[bestAlignmentIdx]

        pDelayPara->SetUtteranceInfo(pResamplingResults[bestAlignmentIdx]->mNumUtterances, pResamplingResults[bestAlignmentIdx]->mpStartSampleUtterance, pResamplingResults[bestAlignmentIdx]->mpStopSampleUtterance, pResamplingResults[bestAlignmentIdx]->mpDelayUtterance)

        OTA_FLOAT FinalRatioApplied=1.0
        OTA_FLOAT FinalRatioMeasured=1.0
        for (int i=0  i<bestAlignmentIdx  i++)
        {
            if (pResamplingResults[i]->mRelSamplerateDev>0)

                FinalRatioMeasured *= (1.0 / (1.0 + (1.0 - pResamplingResults[i]->mRelSamplerateDev)))

        }
```

```
1347        FinalRatioApplied = FinalRatioMeasured
1348        if (pResamplingResults[bestAlignmentIdx]->mRelSamplerateDev>0)
1349
1350            FinalRatioMeasured *= (1.0 / (1.0 + (1.0 -
    pResamplingResults[bestAlignmentIdx]->mRelSamplerateDev)))
1351
1352        PolqaResults->m_MeasuredSamplerate = FinalRatioMeasured*PolqaResults->m_SampleFrequencyHz
1353        PolqaResults->m_AppliedSamplerate  = FinalRatioApplied *PolqaResults->m_SampleFrequencyHz
1354
1355        (*pTaResult)->CopyFrom(pResamplingResults[bestAlignmentIdx])
1356
1357        if (bestAlignmentIdx == TArunIndex)
1358        {
1359            matbCopy(pRefSig[0], pDelayPara->pOriginalSamples,  numRefSamples)
1360            pDelayPara->OriginalNumberOfSamples  = numRefSamples
1361            matbCopy(pDegSig[0], pDelayPara->pDistortedSamples, numDegSamples)
1362            pDelayPara->DistortedNumberOfSamples = numDegSamples
1363        }
1364        else if (bestAlignmentIdx != 0)
1365        {
1366            XFLOAT Ratio = 1.0
1367            for (int i = 0  i < bestAlignmentIdx  i++)
1368            {
1369
1370                {
1371
1372                    Ratio = 1.0 + (1.0 - pResamplingResults[i]->mRelSamplerateDev)
1373
1374                }
1375            }
1376
1377            if (Ratio < 1.0)
1378            {
1379                matConvertSamplerate(pDelayPara->pDistortedSamples,
    pDelayPara->DistortedNumberOfSamples,
1380                                     pDelayPara->pDistortedSamples, pDelayPara->MaxSigLen, Ratio,
    &NewLen)
1381
1382                pDelayPara->DistortedNumberOfSamples = NewLen
1383            }
1384            else
1385            {
1386                matConvertSamplerate(pDelayPara->pOriginalSamples,
    pDelayPara->OriginalNumberOfSamples,
1387                                     pDelayPara->pOriginalSamples, pDelayPara->MaxSigLen, 1.0/Ratio,
    &NewLen)
1388
1389                pDelayPara->OriginalNumberOfSamples = NewLen
1390            }
1391
1392        }
1393
1394    }
1395
1396    for (int i = 0  i < MAX_TA_RUNS  i++)
1397    {
1398        if (pResamplingResults[i]) delete pResamplingResults[i]
1399        pResamplingResults[i] = NULL
1400    }
1401    for (int i = 0  i < 2  i++)
1402    {
1403        matFree(pRefSig[i])
1404        matFree(pDegSig[i])
1405        pRefSig[i] = pDegSig[i] = NULL
1406    }
1407
1408    *pDone = Done
1409
```

```
    if (rc)
        pDelayPara->Check()

    return rc
}

bool DoCalculateDelayDegPlus(CDelayPara* pDelayPara, POLQA_RESULT_DATA* PolqaResults)
{
    bool rc=true

    int i
    long f
    long ClockCycles=0
    double TimeDiff=0
    MAT_HANDLE mh = pDelayPara->mh
    TACheckTimeMatInit(mh, 1)



    ITempAlignment* pTA = CreateAlignment(TA_FOR_SPEECH)
    OTA_RESULT* pTaResult = new OTA_RESULT()

    bool Done = false
    pDelayPara->PitchFreqRef = -1
    pDelayPara->PitchFreqDeg = -1
    do
    {


        if (0 && pDelayPara->LogFile)
        {
            fprintf(pDelayPara->LogFile, "\nDoCalculateDelayDeg() 1\n")
            OTA_FLOAT EnergyRef = matSum(pDelayPara->pOriginalSamples,
pDelayPara->OriginalNumberOfSamples)
            OTA_FLOAT EnergyDeg = matSum(pDelayPara->pDistortedSamples,
pDelayPara->DistortedNumberOfSamples)
            fprintf(pDelayPara->LogFile, "\tSample sum ref:\t%.15e\n", EnergyRef)
            fprintf(pDelayPara->LogFile, "\tSample sum deg:\t%.15e\n", EnergyDeg)
        }

        rc = DoAlignmentPlus(pDelayPara, PolqaResults, pTA, &pTaResult, 0x8, &Done)



        XFLOAT* pRefSig=pDelayPara->pOriginalSamples
        XFLOAT* pDegSig=pDelayPara->pDistortedSamples

        if (rc && Done)
        {
            TACheckTimeMatEval(mh, 1, &ClockCycles, &TimeDiff)
            if (pDelayPara->LogFile)
                fprintf(pDelayPara->LogFile, "\nTime between pTA->Run() and calculating noise
switching indicator: %.3lfs\n", TimeDiff)

            TACheckTimeMatInit(mh, 1)

            ConvertFrameRateOfVectors(pTaResult, pDelayPara, PolqaResults)

            ConvertActivityFlag(pDelayPara->LogFile, pTaResult, pDelayPara, PolqaResults)
            AddProcessingTime(PolqaResults, "Vector conversion", TimeDiff, ClockCycles)

            {

                AlignPitchVectors(pDelayPara, pDelayPara->pPitchVecOfRef,
pDelayPara->pPitchVecOfDeg, pDelayPara->pIgnoreFrameFlags)

                pDelayPara->PitchFreqDeg = AveragePitchFrequencyFromPitchVector(pDelayPara,
pDelayPara->pPitchVecOfDeg)
```

```
1473                pDelayPara->PitchFreqRef = AveragePitchFrequencyFromPitchVector(pDelayPara,
    pDelayPara->pPitchVecOfRef)
1474                pDelayPara->PitchFrameSize = pTA->GetPitchFrameSize()
1475
1476                TACheckTimeMatEval(mh, 1, &ClockCycles, &TimeDiff)
1477                if (pDelayPara->LogFile)
1478                    fprintf(pDelayPara->LogFile, "\nTime required to align pitch information:
    %.3lfs\n", TimeDiff)
1479             }
1480         }
1481
1482         if (rc && Done && pTaResult->mNumUtterances && abs(pTaResult->mpDelayUtterance[0]) >
    0.8*(((pDelayPara->DistortedNumberOfSamples) < (pDelayPara->OriginalNumberOfSamples)) ?
    (pDelayPara->DistortedNumberOfSamples) : (pDelayPara->OriginalNumberOfSamples)))
1483         {
1484
1485             TACheckTimeMatInit(mh, 1)
1486             long RequiredOffset=0
1487             if (pTaResult->mpDelayUtterance[0]<0)
1488             {
1489                 RequiredOffset = pTaResult->mpDelayUtterance[0]
1490                 if (pDelayPara->LogFile)
1491                     fprintf(pDelayPara->LogFile, "\n\n***** Shifting degraded signal by %ld samples
    left\n\n", RequiredOffset)
1492                 unsigned long NewLen=pDelayPara->DistortedNumberOfSamples+RequiredOffset
1493                 pDelayPara->DistortedNumberOfSamples = NewLen
1494                 for (i=0  i<NewLen  i++)
1495                     pDegSig[i] = pDegSig[i-RequiredOffset]
1496                 for (i=0  i<pDelayPara->pDelayUtterance->GetSize()  i++)
1497                 {
1498                     pDelayPara->pDelayUtterance->m_pData[i] -= RequiredOffset
1499                     pDelayPara->pStartSampleUtterance->m_pData[i] += RequiredOffset
1500                     pDelayPara->pStopSampleUtterance->m_pData[i] += RequiredOffset
1501                 }
1502
1503                 for (i=0  i<pDelayPara->pDelayUtterance->GetSize()  i++)
1504                 {
1505                     if (pDelayPara->pStopSampleUtterance->m_pData[i]<0)
1506                     {
1507                         DeleteUtteranceFromVector(i, pDelayPara)
1508                         pTaResult->mNumUtterances--
1509                     }
1510                     if (pDelayPara->pStartSampleUtterance->m_pData[i]<0)
1511                     {
1512                         pDelayPara->pStartSampleUtterance->m_pData[i] = 0
1513                     }
1514                 }
1515
1516                 const int OffsetInFrames = (RequiredOffset+pDelayPara->Framesize/2) /
    pDelayPara->Framesize
1517                 if (OffsetInFrames)
1518                 {
1519                     for (i=0  i<pDelayPara->MaxModelFrames+OffsetInFrames  i++)
1520                     {
1521                         pDelayPara->pPitchVecOfRef[i] = pDelayPara->pPitchVecOfRef[i-OffsetInFrames]
1522                         pDelayPara->pPitchVecOfDeg[i] = pDelayPara->pPitchVecOfDeg[i-OffsetInFrames]
1523                         pDelayPara->pActiveFrameFlags[i] =
    pDelayPara->pActiveFrameFlags[i-OffsetInFrames]
1524                         pDelayPara->pIgnoreFrameFlags[i] =
    pDelayPara->pIgnoreFrameFlags[i-OffsetInFrames]
1525                     }
1526                     pDelayPara->MaxModelFrames += OffsetInFrames
1527
1528                     for (i=0  i<PolqaResults->m_NumberOfFrames+OffsetInFrames  i++)
1529                     {
1530                         PolqaResults->m_DelayReliabilityPerFrame[i] =
    PolqaResults->m_DelayReliabilityPerFrame[i-OffsetInFrames]
1531
```

```
1532                        PolqaResults->m_DelayPerFrame[i] =
    PolqaResults->m_DelayPerFrame[i-OffsetInFrames] - RequiredOffset
1533                    }
1534                    PolqaResults->m_NumberOfFrames+=OffsetInFrames
1535                }
1536            }
1537            else
1538            {
1539                RequiredOffset = -pTaResult->mpDelayUtterance[0]
1540                if (pDelayPara->LogFile)
1541                    fprintf(pDelayPara->LogFile, "\n\n***** Shifting original signal by %ld samples
    left\n\n", RequiredOffset)
1542                unsigned long NewLen=pDelayPara->OriginalNumberOfSamples+RequiredOffset
1543                pDelayPara->OriginalNumberOfSamples = NewLen
1544                for (i=0  i<NewLen  i++)
1545                    pRefSig[i] = pRefSig[i-RequiredOffset]
1546                for (i=0  i<pTaResult->mNumUtterances  i++)
1547                    (*pDelayPara->pDelayUtterance).m_pData[i] += RequiredOffset
1548
1549                for (i=0  i<pDelayPara->MaxModelFrames  i++)
1550                    PolqaResults->m_DelayPerFrame[i] += RequiredOffset
1551
1552                int OffsetInFrames = (RequiredOffset+pDelayPara->Framesize/2) /
    pDelayPara->Framesize
1553                pDelayPara->MaxModelFrames += OffsetInFrames
1554            }
1555            TACheckTimeMatEval(mh, 1, &ClockCycles, &TimeDiff)
1556            if (pDelayPara->LogFile)
1557                fprintf(pDelayPara->LogFile, "\nTime required to fix very long offsets: %.3lfs\n",
    TimeDiff)
1558        }
1559
1560        if (rc && Done)
1561        {
1562
1563            XFLOAT AvgDelay=0
1564            for (f=0  f<pTaResult->mNumFrames  f++)
1565                AvgDelay += -pTaResult->mpDelay[f]
1566            AvgDelay /= (XFLOAT)(pTaResult->mNumFrames)
1567
1568            PolqaResults->m_MinDelay=100000000.0f
1569            PolqaResults->m_MaxDelay=-100000000.0f
1570            for (f=0  f<pTaResult->mNumFrames  f++)
1571            {
1572                XFLOAT val=-pTaResult->mpDelay[f]
1573                if(val>PolqaResults->m_MaxDelay)PolqaResults->m_MaxDelay=val
1574                if(val<PolqaResults->m_MinDelay)PolqaResults->m_MinDelay=val
1575            }
1576
1577            PolqaResults->m_GlobalDelay = AvgDelay/PolqaResults->m_SampleFrequencyHz
1578            PolqaResults->m_CrudeDelay = AvgDelay
1579            PolqaResults->m_SNRDegdB = pTaResult->mSNRDegdB
1580            PolqaResults->m_SNRRefdB = pTaResult->mSNRRefdB
1581            PolqaResults->m_SignalLevelRef = pTaResult->mSignalLevelRef
1582            PolqaResults->m_SignalLevelDeg = pTaResult->mSignalLevelDeg
1583            PolqaResults->m_NoiseLevelRef = pTaResult->mNoiseLevelRef
1584            PolqaResults->m_NoiseLevelDeg = pTaResult->mNoiseLevelDeg
1585            PolqaResults->m_NoiseThresholdRef = pTaResult->mNoiseThresholdRef
1586            PolqaResults->m_NoiseThresholdDeg = pTaResult->mNoiseThresholdDeg
1587        }
1588
1589        pTA->Free()
1590
1591    }   while (!Done && rc)
1592
1593    if (rc) DumpAlignedFile(pDelayPara, "d:\\temp\\Aligned.OptInterface.pcm")
1594
1595    delete pTaResult
```

```
1596      pTA->Destroy()
1597
1598
1599
1600
1601      return rc
1602  }
1603
1604  }
1605
```