```
1
2
3       typedef double XFLOAT
4       typedef double OTA_FLOAT
5
6   using  std
7
8
9   {
10
11  PreAlignment::PreAlignment(XFLOAT const* refBuff, long numRefSamples, XFLOAT const* degBuff, long
    numDegSamples,
12                             int appType, int samplingFreq, char const* outputFile, FILE* pLogFile,
    void* mh)
13  : mFNameRef            (NULL),
14    mFNameDeg            (NULL),
15    mFNameOutput         (outputFile),
16    mAppType             ((appType%100) % 10),
17    mRef                 (NULL),
18    mDeg                 (NULL),
19    mAligner             (NULL),
20    mResamplingResult    (NULL),
21    mSegListTraversal    (NULL),
22    mSamplingFreq        (samplingFreq),
23    mMaxSigLenBuff       (NULL),
24    matHandle            (NULL),
25    mpLogFile            (NULL)
26  {
27      OPTTRY
28      {
29          if (mh == NULL)
30              OPTTHROW(( string("MathLib-Handle == NULL.")))
31
32          matHandle = (MAT_HANDLE)mh
33          mpLogFile = pLogFile
34
35          if (mAppType < kNarrowBand || mAppType > kSuperWideBand)
36              OPTTHROW(( SQError(SQERR_UNSUPPORTED_APPTYPE,"Invalid application type in input
    parameters.")))
37          if (samplingFreq < MIN_SAMPLING_RATE || samplingFreq > MAX_SAMPLING_RATE)
38              OPTTHROW(( SQError(SQERR_SAMPLING_FREQ,"Unsupported sampling frequency for reference
    and/or degraded file.")))
39
40          int iRetVal = ReadData(refBuff, numRefSamples, degBuff, numDegSamples)
41          if (iRetVal < 0)
42              OPTTHROW((  SQError(iRetVal, "Error while reading data.")))
43
44          iRetVal = Run()
45          if (iRetVal != SQ_NO_ERRORS)
46              OPTTHROW((  SQError(iRetVal, "Error PreAlignment time-domain calculations.")))
47      }
48
49      OPTCATCH ((SQError err))
50      {
51          delete mRef
52          delete mDeg
53          delete mAligner
54          delete mResamplingResult
55          delete mSegListTraversal
56          matFree(mMaxSigLenBuff)
57          OPTTHROW ((SQError(err.ErrCode(),
58              "Could not start PreAlignment algorithm.\nThe following error message was returned: "
59              + err.ErrMsg() + "\n")))
60
61      }
62      OPTCATCH( (...) )
63      {
64          delete mRef
```

```
        delete mDeg
        delete mAligner
        delete mResamplingResult
        delete mSegListTraversal
        matFree(mMaxSigLenBuff)
        OPTTHROW (( SQError(SQERR_OTHER, "Could not start PreAlignment algorithm because of an unknown error.\n") ))
    }
}

PreAlignment::~PreAlignment()
{
    delete   mRef
    delete   mDeg
    delete   mAligner
    delete   mResamplingResult
    delete   mSegListTraversal
    matFree(mMaxSigLenBuff)

    mRef                = mDeg      = NULL
    mAligner            = NULL
    mResamplingResult   = NULL
    mSegListTraversal   = NULL
    mMaxSigLenBuff      = NULL
}

TA_SegList const* PreAlignment::GetMergedSegList() const
{
    if (mAligner == NULL)
        return NULL

    return mAligner->MergedSegments()
}

XFLOAT PreAlignment::GetResamplingFac() const
{
    if (mAligner == NULL || mResamplingResult == NULL)
        return (XFLOAT)(-1.0)

    return mResamplingResult->fMeanResamplingFac
}

XFLOAT PreAlignment::GetDegSNR() const
{
    if (mAligner == NULL)
        return (XFLOAT)(-1.0)

    return mAligner->SNRDeg()
}

XFLOAT PreAlignment::GetMatchQuality() const
{
    if (mAligner == NULL)
        return (XFLOAT)(-1.0)

    return mAligner->MatchQuality()
}

bool PreAlignment::ExtremeMatchFound() const
{
    if (mAligner == NULL)
        return false

    return mAligner->ExtremeMatchFound()
}

int PreAlignment::Run()
{
```

```
132     stringstream errorStream
133     clock_t startTime = clock()
134     int iRetVal = SQ_NO_ERRORS
135
136     mTraversalVec.clear()
137
138     iRetVal = SanityCheck()
139     if (iRetVal < 0)
140     {
141         switch(iRetVal)
142         {
143         case SQERR_CORRUPTED_FILE:    errorStream << "Reference and/or degraded file(s) too short. They may have not been properly recorded.\n"  break
144         case SQERR_REF_FILE_TOO_LONG: errorStream << "Reference file is too long. It should be shorter than " << MAX_SPEECH_DURATION << " seconds.\n"  break
145         case SQERR_DEG_FILE_TOO_LONG: errorStream << "Degraded file is too long. It should be shorter than " << MAX_SPEECH_DURATION << " seconds.\n"  break
146         default:                      errorStream << "Unknown error.\n"  break
147         }
148         OPTTHROW ((SQError(iRetVal, errorStream.str())))
149
150     }
151
152     if (iRetVal == SQ_NO_ERRORS)
153         iRetVal = Preprocess()
154
155     if (iRetVal == SQ_NO_ERRORS)
156         iRetVal = AlignSignals()
157
158     return iRetVal
159 }
160
161 TraversalVecType const& PreAlignment::TraversalVec() const
162 {
163     return mTraversalVec
164 }
165
166 int PreAlignment::ReadData (XFLOAT const *refBuff, long numRefSamples,
167                             XFLOAT const *degBuff, long numDegSamples)
168 {
169     OPTTRY
170     {
171         if(numRefSamples > 0 && numDegSamples > 0)
172         {
173             mRef = new SQSignal(refBuff, numRefSamples, mSamplingFreq, STD_BIT_RESOLUTION)
174             mDeg = new SQSignal(degBuff, numDegSamples, mSamplingFreq, STD_BIT_RESOLUTION)
175         }
176         else
177             OPTTHROW( SQError(SQERR_CORRUPTED_FILE))
178     }
179     OPTCATCH ((SQError err))
180     {
181         OPTTHROW( SQError(err.ErrCode(), "ERROR in ReadData: " + err.ErrMsg()))
182     }
183     OPTCATCH((string errorMsg))
184     {
185         OPTTHROW(SQError(SQERR_READDATA, "ERROR in ReadData: " + errorMsg))
186     }
187     OPTCATCH((...))
188     {
189         OPTTHROW(SQError(SQERR_READDATA, "ERROR in ReadData: Unknown error.\n"))
190     }
191
192     return SQ_NO_ERRORS
193 }
194
195 int PreAlignment::SanityCheck()
196 {
```

```
197     int retVal = SQ_NO_ERRORS
198
199     if(mRef->NrOfSamples() / (XFLOAT)mRef->SamplingFreq() < (XFLOAT)2.2)
200         retVal = SQERR_CORRUPTED_FILE
201
202     if(mRef->NrOfSamples() / (XFLOAT)mRef->SamplingFreq() > MAX_SPEECH_DURATION)
203         retVal = SQERR_REF_FILE_TOO_LONG
204
205     if(mDeg->NrOfSamples() / (XFLOAT)mDeg->SamplingFreq() < MIN_SPEECH_DURATION)
206         retVal = SQERR_CORRUPTED_FILE
207
208     if(mDeg->NrOfSamples() / (XFLOAT)mDeg->SamplingFreq() > MAX_SPEECH_DURATION)
209         retVal = SQERR_DEG_FILE_TOO_LONG
210
211     return retVal
212 }
213
214 int PreAlignment::Preprocess()
215 {
216     int iRetVal = SQ_NO_ERRORS
217
218     if (mRef == NULL || mDeg == NULL)
219         OPTTHROW ((SQError(SQERR_PREPROC, "Cannot perform preprocessing because reference and/or
    degraded signals were not created.")))
220     if (mRef->IsInvalidSignal() || mDeg->IsInvalidSignal())
221         OPTTHROW ((SQError(SQERR_PREPROC, "Cannot perform preprocessing because reference and/or
    degraded signal contain invalid data.")))
222     OPTTRY
223     {
224         int maxSamplingFreq = max(max(mRef->SamplingFreq(), TA_SAMPLING_RATE), mDeg->SamplingFreq())
225
226         int maxSigLen = max(1024, (int)ceil(
227             max(mRef->NrOfSamples()/(XFLOAT)mRef->SamplingFreq()*maxSamplingFreq + 1,
228                 mDeg->NrOfSamples()/(XFLOAT)mDeg->SamplingFreq()*maxSamplingFreq + 1)))
229         if(mMaxSigLenBuff)
230             matFree(mMaxSigLenBuff)
231         mMaxSigLenBuff = (XFLOAT*)matMalloc(maxSigLen * sizeof(XFLOAT))
232
233         if (mAppType == kSuperWideBand)
234             mDeg->Preprocess(maxSamplingFreq, SQ_SIGNAL_NO_LEVEL_ALIGN,
235                 FRAME_LEN, FRAME_OVERLAP_RATIO, MIN_LEVEL_DB, mAppType, mMaxSigLenBuff, 0, mpLogFile)
236         else
237             mDeg->Preprocess(maxSamplingFreq, REF_ASL_LEVEL,
238                 FRAME_LEN, FRAME_OVERLAP_RATIO, MIN_LEVEL_DB, mAppType, mMaxSigLenBuff, 0, mpLogFile)
239
240         if (mDeg->IsInvalidSignal())
241             return SQERR_PREPROC
242
243         if (mDeg->UnalignedASL() < MIN_ASL_DEG)
244             return SQERR_SPEECH_ACTIVITY
245
246         if ((mDeg->End() - mDeg->Start()) / (XFLOAT)mDeg->SamplingFreq() < MIN_SPEECH_DURATION)
247             return SQERR_DEG_FILE_TOO_SHORT
248
249         mRef->Preprocess(maxSamplingFreq,
250             mDeg->CurrentASL(),
251             FRAME_LEN,
252             FRAME_OVERLAP_RATIO,
253             MIN_LEVEL_DB, mAppType, mMaxSigLenBuff, 0, mpLogFile)
254
255         if (mRef->IsInvalidSignal())
256             return SQERR_PREPROC
257
258         if ((mRef->End() - mRef->Start()) / (XFLOAT)mRef->SamplingFreq() < MIN_SPEECH_DURATION)
259             return SQERR_REF_FILE_TOO_SHORT
260
261     }
262     OPTCATCH ((string errorMsg))
```

```
    {
        OPTTHROW ((SQError(SQERR_PREPROC, "Preprocessing failed: " + errorMsg)))
    }
    OPTCATCH ((SQError err))
    {
        OPTTHROW ((SQError(err.ErrCode(), "Preprocessing failed: " + err.ErrMsg())))
    }
    OPTCATCH ((...))
    {
        OPTTHROW ((SQError(SQERR_PREPROC, "Preprocessing failed: Unknown error.\n")))
    }

    return iRetVal
}

int PreAlignment::AlignSignals()
{
    OPTTRY
    {
        mAligner          = new SQTimeAlignment(*mRef, *mDeg, mMaxSigLenBuff, 1.0, matHandle,
mpLogFile)
        mResamplingResult = new SQTA_ResampResult()

    }
    OPTCATCH ((string errorMsg))
    {
        OPTTHROW ((SQError(SQERR_TIMEALIGNMENT, "Time alignment failed: " + errorMsg)))
    }
    OPTCATCH ((SQError err))
    {
        OPTTHROW ((SQError(err.ErrCode(), "Time alignment failed: " + err.ErrMsg())))
    }
    OPTCATCH ((...))
    {
        OPTTHROW ((SQError(SQERR_TIMEALIGNMENT, "Time alignment failed: unknown error.")))
    }

    return SQ_NO_ERRORS
}

int PreAlignment::TraverseSegList(XFLOAT frameLengthInSec, XFLOAT frameStepInSec)
{
    if (mRef == NULL || mDeg == NULL || mAligner == NULL ||
        mAligner->MergedSegments() == NULL || mAligner->UnusedDegSegments() == NULL)
        OPTTHROW ((string("Cannot call TraverseSegList because previous modules did not execute
successfully.")))

    OPTTRY
    {
        if (frameLengthInSec <= 0.0f || frameStepInSec <= 0.0f || frameLengthInSec < frameStepInSec)
            OPTTHROW ((SQError(SQERR_OTHER, "Invalid frame length/step value(s).")))

        int frameLengthInSamples = round(mRef->SamplingFreq() * frameLengthInSec)
        int frameStepInSamples   = round(mRef->SamplingFreq() * frameStepInSec)
        mTraversalVec.clear()
        mSegListTraversal = new SegListTraversal(mAligner, mResamplingResult, mRef, mDeg,
                                                 frameLengthInSamples, frameStepInSamples)

        int curRefPos, curDegPos, minSearchPos, maxSearchPos
        int expectedDegPos = -1
        FRAMETYPE type
        XFLOAT     reliability
        bool      degActivity
        while (mSegListTraversal->FullTraversal(curRefPos, curDegPos, minSearchPos, maxSearchPos)
               != LEAVE_LOOP)
        {
            if (expectedDegPos < 0 && !mSegListTraversal->RefMissingInDeg())
                expectedDegPos = frameStepInSamples * (int)ceil(curDegPos /
```

```
    (XFLOAT)frameStepInSamples)
329
330             if (mSegListTraversal->SegType() == TA_SEG_MATCHED)
331                 degActivity = true
332             else
333                 degActivity =
    mDeg->VADprofile()[mDeg->SamplePosToFrameNum(curDegPos+frameLengthInSamples/2)] ==
    SQ_VAD_ACT_SPEECH
334
335             if (mSegListTraversal->RefMissingInDeg())
336                 type = MISSING_SPEECH
337             else if (mSegListTraversal->DegMissingInRef())
338                 type = INSERTED_SIG
339             else switch (mSegListTraversal->SegType())
340             {
341                 case TA_SEG_MATCHED: type = FIXED_SPEECH       break
342                 case TA_SEG_GUESSED: type = SEARCHABLE_SPEECH  break
343                 case TA_SEG_PAUSE:   type = PAUSE              break
344                 default: OPTTHROW ((string("Unexpected frame type.\n")))
345             }
346
347             reliability = mSegListTraversal->SegReliability()
348
349             if (expectedDegPos >= 0 && mTraversalVec.size() > 0 &&
350                 curDegPos >= expectedDegPos + frameStepInSamples/2)
351             {
352                 FRAMETYPE prevType = mTraversalVec.back().type
353                 if (prevType != MISSING_SPEECH)
354                 {
355                     if (prevType == PAUSE && (type == SEARCHABLE_SPEECH || type == FIXED_SPEECH))
356                     {
357                         int posDec = expectedDegPos - curDegPos
358                         mTraversalVec.push_back(TraversalStruct(curRefPos+posDec, curDegPos+posDec,
359                                                                 minSearchPos+posDec,
    maxSearchPos+posDec,
360                                                                 0.0f, degActivity,
    SEARCHABLE_SPEECH))
361                     }
362                     else if (prevType == INSERTED_SIG)
363                     {
364                         int posInc = expectedDegPos - mTraversalVec.back().degPos
365                         mTraversalVec.push_back(TraversalStruct(mTraversalVec.back().refPos,
    mTraversalVec.back().degPos+posInc,
366                                                                 mTraversalVec.back().minPos,
    mTraversalVec.back().maxPos,
367                                                                 0.0f,
    mTraversalVec.back().degActivity,
    mTraversalVec.back().type))
368                     }
369                     else
370                     {
371                         int posInc = expectedDegPos - mTraversalVec.back().degPos
372                         int newDelay = mTraversalVec.back().refPos - mTraversalVec.back().degPos
373                         if ((prevType == SEARCHABLE_SPEECH || prevType == FIXED_SPEECH) &&
374                             (type     == SEARCHABLE_SPEECH || type     == FIXED_SPEECH))
375                             newDelay = (newDelay + (curRefPos-curDegPos)) / 2
376                         int newRefPos = mTraversalVec.back().degPos + posInc + newDelay
377                         mTraversalVec.push_back(TraversalStruct(newRefPos,
    mTraversalVec.back().degPos+posInc,
378                                                                 min(mTraversalVec.back().minPos+posIn
    c, newRefPos),
379                                                                 max(mTraversalVec.back().maxPos+posIn
    c, newRefPos),
380                                                                 0.0f,
    mTraversalVec.back().degActivity,
    mTraversalVec.back().type))
381                     }
382                 }
```

```
383                 else
384                 {
385                     int posDec = expectedDegPos - curDegPos
386                     int firstMissingFrameIdx = (int)mTraversalVec.size()-1
387                     for (  firstMissingFrameIdx > 0 && mTraversalVec.at(firstMissingFrameIdx).type ==
    MISSING_SPEECH  firstMissingFrameIdx--)
388                     firstMissingFrameIdx++
389                     mTraversalVec.push_back(TraversalStruct(curRefPos+posDec, curDegPos+posDec,
390                                             min(min(minSearchPos, curRefPos+posDec),
    mTraversalVec.at(firstMissingFrameIdx).refPos),
    maxSearchPos,
391                                             0.0f, degActivity, type == FIXED_SPEECH ?
    SEARCHABLE_SPEECH : type))
392                 }
393
394                 if (mTraversalVec.back().type != MISSING_SPEECH)
395                     expectedDegPos += frameStepInSamples
396
397                 mTraversalVec.push_back(TraversalStruct(curRefPos, curDegPos,
398                     minSearchPos, maxSearchPos,
399                     reliability, degActivity, type))
400
401                 if (!mSegListTraversal->RefMissingInDeg())
402                     expectedDegPos += frameStepInSamples
403             }
404             else if (expectedDegPos >= 0 && mTraversalVec.size() > 0 && type != MISSING_SPEECH &&
405                      curDegPos < expectedDegPos - frameStepInSamples/2)
406             {
407                 FRAMETYPE prevType = mTraversalVec.back().type
408                 if (prevType != MISSING_SPEECH)
409                 {
410                     if (prevType == PAUSE && (type == SEARCHABLE_SPEECH || type == FIXED_SPEECH))
411                     {
412                         mTraversalVec.back().minPos = min(mTraversalVec.back().minPos, minSearchPos)
413                         mTraversalVec.back().maxPos = max(mTraversalVec.back().maxPos, maxSearchPos)
414                         mTraversalVec.back().type = SEARCHABLE_SPEECH
415                         mTraversalVec.back().degActivity = mTraversalVec.back().degActivity ||
    degActivity
416                         mTraversalVec.back().reliability = 0.0f
417                     }
418                     else if (prevType == INSERTED_SIG)
419                     {
420                         mTraversalVec.back().minPos = min(mTraversalVec.back().minPos, minSearchPos)
421                         mTraversalVec.back().maxPos = max(mTraversalVec.back().maxPos, maxSearchPos)
422                         mTraversalVec.back().degActivity = mTraversalVec.back().degActivity ||
    degActivity
423                         mTraversalVec.back().reliability = 0.0f
424                     }
425                     else
426                     {
427                         if (type != MISSING_SPEECH)
428                         {
429                             mTraversalVec.back().minPos = min(mTraversalVec.back().minPos,
    minSearchPos)
430                             mTraversalVec.back().maxPos = max(mTraversalVec.back().maxPos,
    maxSearchPos)
431                         }
432                         mTraversalVec.back().degActivity = mTraversalVec.back().degActivity ||
    degActivity
433                         mTraversalVec.back().reliability = 0.0f
434                         if (prevType == FIXED_SPEECH)
435                             mTraversalVec.back().type = SEARCHABLE_SPEECH
436                     }
437                 }
438                 else
439                 {
440                     mTraversalVec.pop_back()
441                     mTraversalVec.push_back(TraversalStruct(curRefPos, curDegPos,
```

```
                                                        minSearchPos, maxSearchPos,
                                                        0.0f, degActivity, type))
                }

            }
            else
            {
                mTraversalVec.push_back(TraversalStruct(curRefPos, curDegPos,
                                                        minSearchPos, maxSearchPos,
                                                        reliability, degActivity, type))

                if (!mSegListTraversal->RefMissingInDeg())
                    expectedDegPos += frameStepInSamples
            }

            mSegListTraversal->MoveToNextFramePair()
        }

        delete mSegListTraversal
        mSegListTraversal = NULL

        XFLOAT SNR = mDeg->CurrentASL() - mDeg->CurrentNoiseLevel()
        int maxIntervalLen = limit(round((0.12*SNR - 1.4)*0.021333/frameStepInSec),
                                   1, round(4 * 0.021333/frameStepInSec))
        int silIntervStart = -1
        for (int i = 1  i < (int)mTraversalVec.size()-1  i++)
        {
            if (mTraversalVec[i].type == PAUSE)
            {
                silIntervStart = -1
                continue
            }

            if (silIntervStart < 0 && mTraversalVec[i].degActivity == false)
                silIntervStart = i
            else if (silIntervStart > 0 && mTraversalVec[i].degActivity == true)
            {
                if (i-silIntervStart <= maxIntervalLen)
                    for (int j = silIntervStart  j < i  j++)
                        mTraversalVec[j].degActivity = true
                silIntervStart = -1
            }
        }

    }
    OPTCATCH ((string errorMsg))
    {
        delete mSegListTraversal
        mSegListTraversal = NULL
        OPTTHROW ((SQError(SQERR_CORE, "Segment list traversal module failed: " + errorMsg)))
    }
    OPTCATCH ((SQError err))
    {
        delete mSegListTraversal
        mSegListTraversal = NULL
        OPTTHROW ((SQError(err.ErrCode(), "Segment list traversal module failed: " + err.ErrMsg())))
    }
    OPTCATCH ((...))
    {
        delete mSegListTraversal
        mSegListTraversal = NULL
        OPTTHROW ((SQError(SQERR_CORE, "Segment list traversal module failed: Unknown error.\n")))
    }

    return SQ_NO_ERRORS
}

}
```

510